Addendum to Reference Manual
Version 2.0 and 8087 Supplement

BORLAND INTERNATIONAL

Borland International
4113 Scotts Valley Drive
Scotts Valley, California 95066

Turbo Pascal

Version 2.0 and 8087 Supplement

Addendum to Reference Manual

BORLAND INTERNATIONAL Inc.
4113 Scotts Valley Drive
Scotts Valley, CA 95066

First edition, April 1984
Printed in the United States of America
9 8 7 6 5 4 3 2 1

Version 2.0 Supplement

RETURN THIS LICENSE AGREEMENT TO BORLAND INTERNATIONAL.

Program License agreement

On the condition that you sign and return this license agreement to Borland International Inc., you are granted a non exclusive and non transferable license to use TURBO Pascal on one CPU only. Failure to sign this agreement and still use the software is illegal.

The license includes the right to develop programs with TURBO Pascal for the purpose of re-sale. Although the TURBO run-time library remains the property of Borland Inc., and is copyrighted by Borland Inc., no royalty shall be payable to Borland Inc. for programs developed with the TURBO Pascal system.

TURBO Pascal remains the property of Borland International Inc. and giving away or selling copies of TURBO Pascal is theft of Borland's property and will be prosecuted to the fullest extent of the law by the lawyers of Borland International.

The Reference Manual is, of course, your property but copyrighted by Borland International Inc.

Borland warrants that all material furnished by Borland constitutes an accurate original manufacture and will replace any such material proven to be defective, provided that such defect is found and reported *within ten days after purchase*. Borland makes no other express or implied warranties with regard to performance or accuracy of TURBO Pascal and pertaining documentation and specifically disclaims any implied warranties of fitness for any particular purpose. Borland shall not be held responsible for any consequential damages that you may possibly incur through the use of TURBO Pascal, whether through Borland's negligence or not.

Termination of License. Any breach of one or more of the provisions of this agreement shall constitute an immediate termination of this agreement. Nevertheless, I agree that in the event of such termination, all provisions of this agreement which protect the rights of Borland shall remain in force.

I hereby acknowledge that I have read this agreement, understand it and agree to be bound by its terms and conditions.

Name and signature:________________________________

Address: ________________________________

________________ My serial no: ________________

Cut Here

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

This addendum describes the following new features introduced in TURBO Pascal version 2 but not yet included in the Reference Manual:

1) **Overlay system.** The overlay system allows you to write programs which are larger than the memory available for program code. This provides for easy implementation of even very large programs.

2) **Dynamic heap.** Standard Pascal's **dispose** procedure is fully implemented, supplementing the more restricted **mark** and **release** procedures of the TURBO version 1.

3) **Additional editor commands.**

   IBM PC and compatibles only:

4) **Colors.**

5) **Graphics.**

6) **Windows.**

7) **Sound.**

These new features are described in detail in the following chapters. The TURBO Pascal Reference Manual applies in full to version 2.

8) Optional 8087 Support

Describes the special 16-bit version of **TURBO** Pascal which uses the optional 8087 coprocessor for *Real* arthmetic. Available now for an additional charge. Call us to order!

Notes:

1. OVERLAY SYSTEM

The overlay system lets you create programs much larger than can be accommodated by the computer's memory. The technique is to collect a number of subprograms (procedures and functions) in one or more files separate from the main program file, which will then at runtime be loaded automatically one at a time into the **same** area in memory.

The following drawing shows a program using one overlay file with five overlay subprograms collected into one **overlay group**, thus sharing the same memory space in the main program:

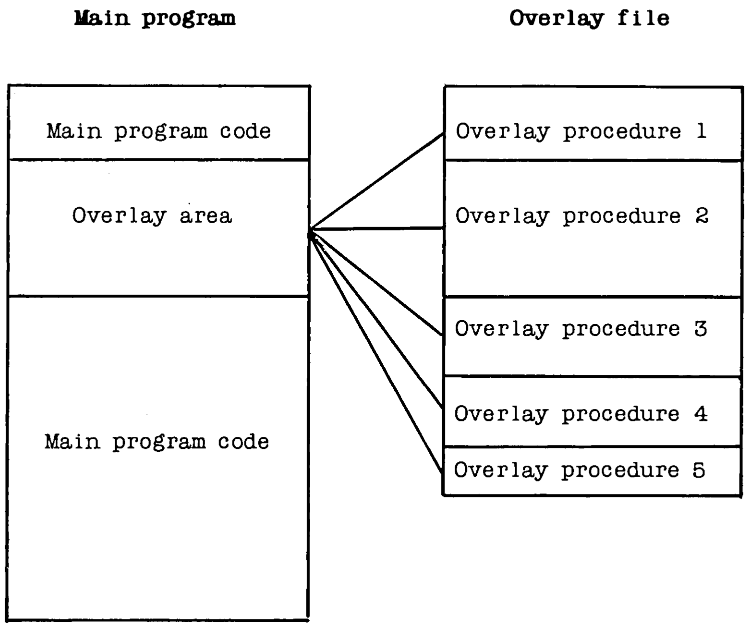

Figure 1-1: Principle of Overlay System

When one of the overlay procedures is called, it is automatically loaded into the overlay area reserved in the main program. This 'gap' is large enough to accommodate the largest of the overlays in the group. The space required by the main program is thus reduced by roughly the sum of all subprograms in the group less the largest of them.

In the example above, overlay procedure 2 is the largest of the five procedures and thus determines the size of the overlay area in the main code. When it is loaded into memory, it occupies the entire overlay area:

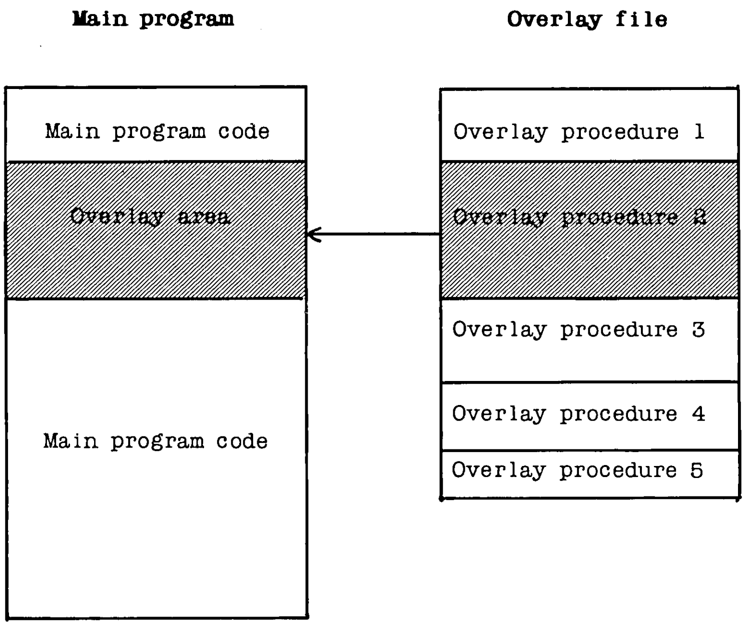

Figure 1-2: Largest Overlay Subprogram Loaded

The smaller subprograms are loaded into the same area of memory, each starting at the first address of the overlay area. Obviously they occupy only part of the overlay area; the remainder is unused:

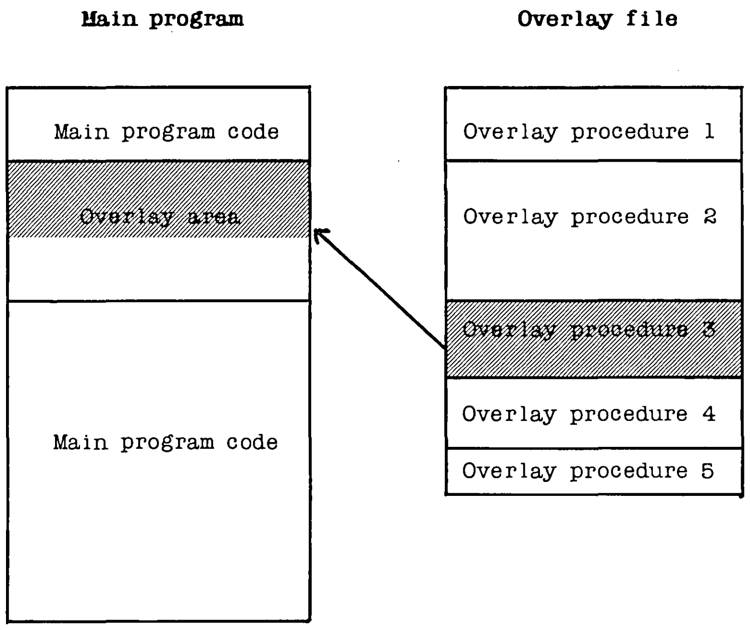

Figure 1-3: Smaller Overlay Subprogram Loaded

As procedures 1, 3, 4, and 5 execute in the same space as used by procedure 2, it is clear that they require no additional space in the main program. There could be many more overlay procedures in this group of overlays; in fact the total size of the overlay procedures could substantially exceed the size of the main program. And they would still require only the space occupied by the largest of them.

The trade off for this extra room for program code is the addition of disk access time each time a procedure is read in from the disk. With good planning, as discussed in section 1.5 , this time is negligible.

1.1 Creating Overlays

Overlay subprograms are created automatically, simply by adding the reserved word **overlay** to the declaration of any procedure or function, e.g.:

```
   overlay procedure Initialize;
```
and
```
   overlay function TimeOfDay: Time;
```

When the compiler meets such a declaration, code is no longer output to the main program file, but to a separate overlay file. The name of this file will be the same as that of the main program, and the type will be a number designating the overlay group, ranging from 000 through 099.

Consecutive overlay subprograms will be grouped together. In other words, as long as overlay subprograms are not separated by any other declaration, they belong to the same group and are placed in the same overlay file.

Example 1:
```
   overlay procedure One;
   begin
     :
   end;

   overlay procedure Two;
   begin
     :
   end;

   overlay procedure Three;
   begin
     :
   end;
```

These three overlay procedures will be grouped together and placed in the same overlay file. If they are the first group of overlay subprograms in a program, the overlay file will be no. *000*.

The three overlay procedures in the following example will be placed in consecutive overlay files, e.g. *.000* and *.001*, because of the declaration of a non-overlay procedure *Count* separating overlay procedures *Two* and *Three*. The separating declaration could be any declaration, e.g. a dummy **type** declaration, if a separation of overlay areas is to be forced.

Example 2:

```
overlay procedure One;
begin
   :
end;

overlay procedure Two;
begin
   :
end;

procedure Count;
begin
   :
end

overlay procedure Three;
begin
   :
end;
```

A separate overlay area is reserved in the main program code for each group of overlay subprograms. Example 2 would thus create the following files:

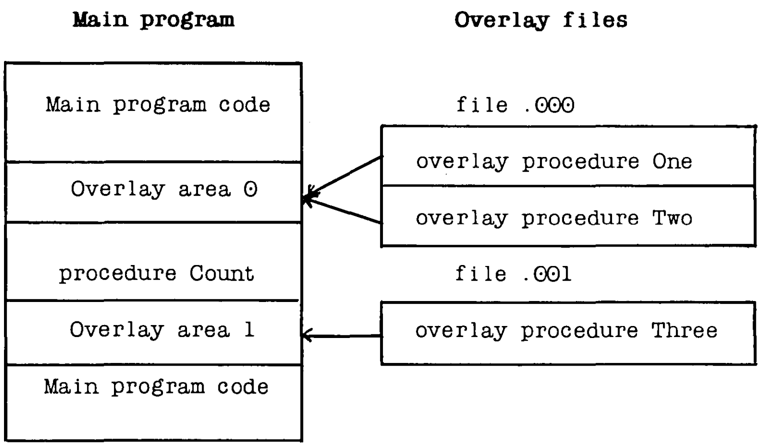

Figure 1 -4: Multiple Overlay Files

Creating an overlay file with only one overlay is meaningless, of course, and is shown here only to illustrate multiple overlay areas.

1.2 Nested Overlays

Overlay subprograms may be nested. This means that an overlay subprogram may itself contain overlay subprograms which may contain overlay sub-programs, etc.

Example 3:

```
program OverlayDemo;
 :
 :
overlay procedure One;
begin
   :
end;

overlay procedure Two;
  overlay procedure Three;
  begin
     :
  end;
begin
   :
end;
 :
 :
```

In this example, two overlay files will be created. File *.000* contains overlay procedures *One* and *Two*, and an overlay area is reserved in the main program to accommodate the largest of these. Overlay file *.001* contains overlay procedure *Three* which is local to overlay procedure *Two*, and an overlay area is created in the code of overlay procedure *Two*:

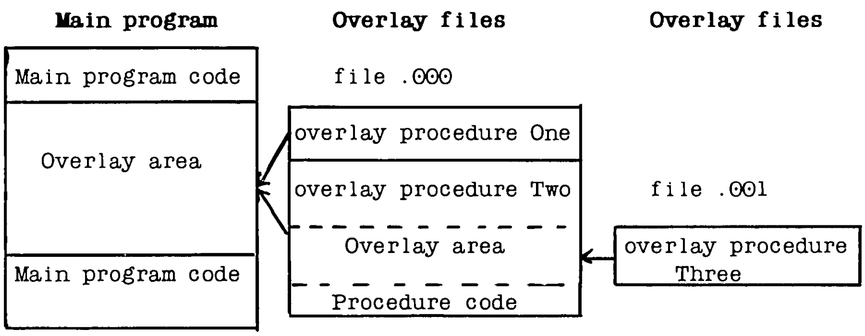

Figure 1-5: Nested Overlay Files

1.3 Automatic Overlay Management

An overlay subprogram is loaded into memory only when called. On each call to an overlay subprogram, a check is first made to see if that subprogram is already present in the overlay area. If not, it will automatically be read in from the appropriate overlay file.

1.4 Placing Overlay Files

During compilation, overlay files will be placed on the logged drive, i.e. on the same drive as the main program file (.COM or .CMD file).

During execution, the system normally expects to find its overlay files on the logged drive. The **O** compiler directive may change this default value. The directive must be followed by a drive letter from A through P indicating specific drives, or an '@', indicating the logged drive. When placed before the first subprogram in a group of subprograms, it affects this and all subsequent overlay groups, until a new **O**-directive is met.

Example 4:

```
{$OL}
overlay function TimeOfYear: Date;
begin
   :
end
:
:
{$O@}
overlay function GetStatus: Stat;
begin
   :
end
```

In this example the first overlay file, starting with the function *TimeOfYear*, must be placed on the *L*-drive at run-time, as must possible subsequent overlay files until the {$O@} directive is met. The overlay file starting with the function *GetStatus* must thus be on the logged drive.

1.5 Efficient Use of Overlays

The overlay technique, of course, adds overhead to a program by adding some extra code to manage the overlays, and by requiring disk accesses during execution. Overlays, therefore, should be carefully planned.

In order not to slow down execution excessively, an overlay subprogram should not be called too often, or - if one **is** called often - it should at least be called without intervening calls to other subprograms in the same overlay file in order to keep disk accesses at a minimum. The added time will of course vary greatly, depending on the actual disk configuration. A 5 1/4" floppy will add much to the run-time, a hard disk much less, and a RAM-disk, as used by many, very little.

To save as much space as possible in the main program, one group of overlays should contain as many individual subprograms as possible. From a pure space-saving point of view, the more subprograms you can put into a single overlay file, the better. The overlay space used in the main program need only accommodate the largest of these subprograms - the rest of the subprograms have a free ride in the same area of memory. This must be weighed against the time considerations discussed above.

1.6 Restrictions Imposed on Overlays

Overlays may not be used in **M**emory compilation mode, i.e. the **C**om (**C**md) or c**H**n-option must be selected on the **O**ptions menu before compiling a program using overlays.

Overlay subprograms in the same group share the same area in memory and thus cannot be present simultaneously. They must therefore **not** call each other. Consequently, they may share the same data area which further adds to the space saved when using overlays (CP/M-80 version only).

In example 1 of this chapter, none of the procedures may therefore call each other. In example 2, however, overlay procedures *One* and *Two* may call overlay procedure *Three,* and overlay procedure *Three* may call each of the other two, because they are in separate files and consequently in separate overlay areas in the main program.

Overlay subprograms may not be **forward** declared. This restriction is easily circumvented, however, by **forward** declaring an ordinary subprogram which then in turn calls the overlay subprogram.

Overlay subprograms cannot be recursive. This restriction may be circumvented by declaring an ordinary recursive subprogram which then in turn calls the overlay subprogram.

Notes:

2. DYNAMIC HEAP

2.1 Dispose

Standard Pascal's garbage collection procedure *Dispose* has been introduced to supplement the *Mark* and *Release* procedures of TURBO Pascal version 1.

The syntax is: `Dispose(Var);` where *Var* is a pointer variable.

NOTICE that *Dispose* and *Mark/Release* use entirely different approaches to heap management - **and never the twain shall meet!** Any one program must use **either** *Dispose* **or** *Mark/Release* to manage the heap. Mixing them will produce unpredictable results.

Dispose allows dynamic memory used *by a specific pointer variable* to be reclaimed for new use, as opposed to *Mark* and *Release* which releases the entire heap *from the specified pointer variable and upward.*

Suppose you have a number of variables which have been allocated on the heap. The following figure illustrates the contents of the heap and the effect of *Dispose(Var3)* and *Mark(Var3) / Release(Var3):*

```
              Heap             After            After
                              Dispose       Mark/Release

           |--------|        |--------|       |--------|
           |  Var1  |        |  Var1  |       |  Var1  |
           |--------|        |--------|       |--------|
           |  Var2  |        |  Var2  |       |  Var2  |
           |--------|        |--------|       |--------|
           |  Var3  |        |        |       |        |
           |--------|        |--------|       |--------|
           |  Var4  |        |  Var4  |       |        |
           |--------|        |--------|       |--------|
           |  Var5  |        |  Var5  |       |        |
           |--------|        |--------|       |--------|
           |  Var6  |        |  Var6  |       |        |
           |--------|        |--------|       |--------|
HiMem      |  Var7  |        |  Var7  |       |        |
```

Figure 2-1: Using Dispose

After *Disposing* a pointer variable, the heap may thus consist of a number of memory areas in use interspersed by a number of free areas. Subsequent calls to *New* will use these if the new pointer variable fits into the space.

2.2 FreeMem

Syntax: FreeMem;

The *FreeMem* standard procedure is used to reclaim an entire block of space on the heap. It is thus the counterpart of *GetMem*. *FreeMem* is called with two parameters:

```
FreeMem(PVar, I);
```

where *PVar* is any pointer variable, and *I* is an integer expression giving the number of bytes to be reclaimed, which must be **exactly** the number of bytes previously allocated to that variable by *GetMem*:

2.3 MaxAvail

Syntax: MaxAvail;

The *MaxAvail* standard function returns the size of the largest consecutive block of free space on the heap. On 16-bit systems this space is in number of *paragraphs* (16 bytes each); on 8-bit systems it is in bytes. The result is an *Integer*, and if more than 32767 paragraphs/bytes are available, *MaxAvail* returns a negative number. The correct number of free paragraphs/bytes is then calculated as 65536.0 + *MaxAvail*. Notice the use of a real constant to generate a *Real* result, as the result is greater than *MaxInt*.

3. NEW EDITOR COMMANDS

The TURBO editor has been enhanced by the introduction of the following additional *WordStar* compatible commands:

Scroll up Ctrl-Z

Scrolls the entire screen upwards one line at a time. The cursor remains on its line until it reaches the top of the screen.

Scroll down Ctrl-W

Scrolls the entire screen downwards one line at a time. The cursor remains on its line until it reaches the bottom of the screen.

To top of screen Ctrl-Q Ctrl-E

Moves the cursor to the top of the screen.

To bottom of screen Ctrl-Q Ctrl-X

Moves the cursor to the bottom of the screen.

To beginning of block Ctrl-Q Ctrl-B

Moves the cursor to the beginning of a marked and displayed block, i.e. to the position of the *block begin* marker set with Ctrl-K Ctrl-B.

To end of block Ctrl-Q Ctrl-K

Moves the cursor to the end of a marked and displayed block, i.e. to the position of the *block end* marker set with Ctrl-K Ctrl-K.

Block hide/display Ctrl-K Ctrl-H

This command causes the visual marking of a block (dim text) to be alternately switched off and on. Block manipulation commands (copy, move, and write to a file) work only when the block is displayed.

3.1 IBM PC and Compatibles

In addition to the *WordStar* commands, the editing keys of IBM PC keyboard have been implemented. This means that while Ctrl-E, Ctrl-X, Ctrl-S, and Ctrl-D still move the cursor up, down, left, and right, you may also use the arrows on the numeric keypad. The following table provides an overview of available editing keys, their functions, and their *WordStar*-command equivalents:

ACTION	PC-KEY	COMMAND
Character left	Left arrow	Ctrl-S
Character right	Right arrow	Ctrl-D
Word left	Ctrl-left arrow	Ctrl-A
Word right	Ctrl-right arrow	Ctrl-F
Line up	Up arrow	Ctrl-E
Line down	Down arrow	Ctrl-X
Page up	PgUp	Ctrl-R
Page down	PgDn	Ctrl-C
To left on line	Home	Ctrl-Q Ctrl-S
To right on line	End	Ctrl-Q Ctrl-D
To top of page	Ctrl-Home	Ctrl-Q Ctrl-E
To bottom of page	Ctrl-End	Ctrl-Q Ctrl-X
To top of file	Ctrl-PgUp	Ctrl-Q Ctrl-R
To end of file	Ctrl-PgDn	Ctrl-Q Ctrl-C
Insert mode on/off	Ins	Ctrl-V
Delete left character	Del	
Mark block begin	F7	Ctrl-K Ctrl-B
Mark block end	F8	Ctrl-K Ctrl-K
Tab	TAB	Ctrl-I

Table 3-1: IBM PC Keyboard Editing Keys

4. IBM PC GOODIES

This chapter applies to the PC-DOS / MS-DOS versions only, and the functions described can be expected to work on IBM PC and compatibles only! If you have problems on a compatible, it's not as compatible as you thought.

The IBM PC gives you a choice of screen modes, each with its own characteristics. Some display characters, some display graphics, and they all have different capabilities of showing colors. TURBO Pascal v. 2 supports all these screen formats and provides an easy way of using them.

The following screen modes are available:

TextMode	25 lines of 40 or 80 characters
GraphColorMode	320X200 dots color graphics
GraphMode	320X200 dots black & white graphics (color on an RGB monitor)
HiRes	640X200 dots black + one color graphics

TURBO Pascal v. 2 furthermore lets you declare windows anywhere on the screen. When you write in such a window, the window behaves exactly as if you were using the entire screen.

Finally, TURBO Pascal v. 2 contains standard procedures which will let you use the PC's sound capabilities in an easy way.

4.1 Text Mode

In text mode, the PC will display 25 lines of either 40 or 80 characters. The procedure to invoke this mode is named *TextMode* and is called as follows:

```
TextMode;
TextMode(BW40);     BW40 is an integer constant with the value 0
TextMode(BW80       BW80 is an integer constant with the value 2
TextMode(C40)       C40 is an integer constant with the value 1
TextMode(C80)       C80 is an integer constant with the value 3
```

The first example with no parameters invokes the text mode which was active last, or the one that is currently active. The next two examples activate black and white text modes with 40 and 80 characters on each line. The final two examples activate color text modes with 40 and 80 characters on each line. Calling *TextMode* will clear the screen.

4.1.1 Colors

In the color text modes, each character may be chosen to be one of 16 colors, and the background may be one of 8 colors. 'Background' here means the cell immediately surrounding each character; the entire screen consists of 40 or 80 by 25 such cells.

The 16 available colors are referred to by numbers. To make things easier, TURBO Pascal v. 2 includes 16 pre-defined integer constants which may be used to identify colors by names:

Dark colors	Light colors
0: Black	**8**: DarkGray
1: Blue	**9**: LightBlue
2: Green	**10**: LightGreen
3: Cyan	**11**: LightCyan
4: Red	**12**: LightRed
5: Magenta	**13**: LightMagenta
6: Brown	**14**: Yellow
7: LightGray	**15**: White

Table 4-1: Text Mode Color Scale

Characters may be any of these colors, whereas the background may be any of the dark colors. Some monitors however, do not recognize the intensity signal used to create the eight light colors. On such monitors, the light colors will be displayed as their dark equivalents.

The **character** color is selected by calling the *TextColor* standard procedure with one integer parameter specifying the desired color:

`TextColor(1);`	selects blue characters
`TextColor(Yellow);`	selects yellow characters

The characters may be made to blink by adding 16 to the color number. There is a pre-defined constant *Blink* for this purpose:

`TextColor(Red + Blink);`	selects red, blinking characters

The **background** color is selected by calling the *TextBackground* standard procedure with one integer parameter specifying the desired color:

`TextBackground(4);`	selects red background
`TextBackground(Magenta);`	selects magenta background

4.1.2 Cursor Addressing

In text mode, two new functions will tell you where the cursor is positioned on the screen:

```
WhereX;
WhereY;
```

return integers expressing the X and Y coordinates of the current cursor position.

4.2 Graphics Modes

Three graphics modes are supported:

GraphColorMode	320X200 dots color graphics
GraphMode	320X200 dots black & white graphics
HiRes	640X200 dots black + one color graphics

In each of these modes, TURBO Pascal provides standard procedures which will plot points at specified coordinates and draw lines between two coordinates:

```
  Plot(X,Y,Color);
  Draw(X1,Y1,X2,Y2,Color);
```

where *X* and *Y* are integer expressions specifying screen coordinates and *Color* is an integer expression specifying the color used as explained in the following.

The upper, left corner of the screen is coordinate 0,0. X coordinates stretch to the right, Y coordinates downward. *Plot*ting outside the screen is ignored, and *Draw*ing outside the screen results in only the part of the line which falls within the screen being displayed (clipping).

Activating one of the graphics modes will clear the screen. The standard procedure *ClrScr* works only in text mode, so the only way to clear a graphics screen is to activate a graphics mode, possibly the one that's already active.

4.2.1 GraphColorMode

GraphColorMode is a standard procedure which activates the 320X200 dots color graphics screen:

```
  GraphColorMode;
```

giving you X-coordinates between 0 and 319 and Y-coordinates between 0 and 199.

The *Plot* and *Draw* procedures may now be used on this screen, using colors selected from a *Palette*. Four such palettes exist, each containing three colors (1-3) and a fourth color (0) which is always equal to the background color (see later):

Color number:	0	1	2	3
Palette 0	Background	Green	Red	Brown
Palette 1	Background	Cyan	Magenta	LightGray
Palette 2	Background	LightGreen	LightRed	Yellow
Palette 3	Background	LightCyan	LightMagenta	White

Table 4-2: Color Palettes in Color Graphics

A palette is activated by a call to the standard procedure *Palette* with a parameter specifying the number of the palette:

`Palette(0)`	activates palette no. 0

Subsequent *Plots* and *Draws* must specify one of the color numbers 0 through 3 which will cause points and lines to be drawn in the colors of the active palette:

`Plot(X,Y,2)`	will plot a red point when palette 0 is active.
`Plot(X,Y,3)`	will plot a yellow point when palette 2 is active.
`Plot(X,Y,0)`	will plot a point in the active background color, in effect erasing that point.

Once a drawing is on the screen, a change of palette will cause all colors on the screen to change to the colors of the new palette. Only three colors plus the color of the background may thus be displayed at one time.

The background (i.e. the entire screen) may be set to any of 16 colors by calling the standard procedure *GraphBackground* with an integer parameter in the range 0 through 15:

`GraphBackground(0);`	sets the screen to black
`GraphBackground(11);`	sets the screen to light cyan

The following color numbers and pre-defined constants are available:

Dark colors	Light colors
0: Black	**8**: DarkGray
1: Blue	**9**: LightBlue
2: Green	**10**: LightGreen
3: Cyan	**11**: LightCyan
4: Red	**12**: LightRed
5: Magenta	**13**: LightMagenta
6: Brown	**14**: Yellow
7: LightGray	**15**: White

Table 4-3: Graphics Background Color Scale

Some monitors do not recognize the intensity signal used to create the eight light colors. On such monitors, the light colors will be displayed as their dark equivalents.

4.2.2 GraphMode

GraphMode is a standard procedure which activates the 320X200 dots black and white graphics screen:

```
GraphMode;
```

giving you X-coordinates between 0 and 319 and Y-coordinates between 0 and 199.

On an **RGB** monitor like the **IBM** Color Display, however, even this mode displays colors. *GraphBackground* works as discussed above, and *Palette* gives you access to the following palette:

Color number:	0	1	2	3
Palette 0	Background	Blue	Red	LightGray
Palette 1	Background	LightBlue	LightRed	White

Table 4-4: Color Palettes in B/W Graphics

Plot and *Draw* work as discussed above.

4.2.3 HiRes

HiRes is a standard procedure which activates the 640X200 dots high resolution graphics screen:

```
    HiRes;
```

giving you X-coordinates between 0 and 639 and Y-coordinates between 0 and 199.

In high resolutions graphics, the background (screen) is always black, and you *Plot* or *Draw* in one color set by the *HiResColor* standard procedure:

`HiResColor(7);`	selects light gray
`HiResColor(Blue);`	selects blue

Changing *HiResColor* causes anything already on the screen to change to the new color.

The one color may be chosen from the following 16 colors:

Dark colors	**Light colors**
0: Black	**8**: DarkGray
1: Blue	**9**: LightBlue
2: Green	**10**: LightGreen
3: Cyan	**11**: LightCyan
4: Red	**12**: LightRed
5: Magenta	**13**: LightMagenta
6: Brown	**14**: Yellow
7: LightGray	**15**: White

Table 4-5: High Resolution Graphics Color Scale

Some monitors do not recognize the intensity signal used to create the eight light colors. On such monitors, the light colors will be displayed as their dark equivalents.

In high resolution graphics, *Plot*s and *Draw*s must specify color numbers 1 or 0 which will cause points and lines to be drawn in the active color or in the background color:

`Plot(X,Y,1)`	will plot a red point when HiResColor 4 is active, yellow when HiResColor 14 is active, etc.
`Plot(X,Y,0)`	will plot a point in the background color, black, in effect erasing that point.

4.3 Windows

When you are in text mode, the *Window* procedure allows you to define any area on the screen as the active window:

```
Window(X1,Y1,X2,Y2);
```

where X_1 and Y_1 are the coordinates of the upper left corner of the window, X_2 and Y_2 are the lower right corner coordinates.

The default window is `Window(1,1,80,25);` in 80-column modes and `Window(1,1,40,25);` in 40-column modes, i.e. the entire screen.

Screen coordinates are always relative to the active window. This means that the statement:

```
Window(20,8,60,17);
```

defines the center portion of the physical screen to be your entire window - or 'logical screen':

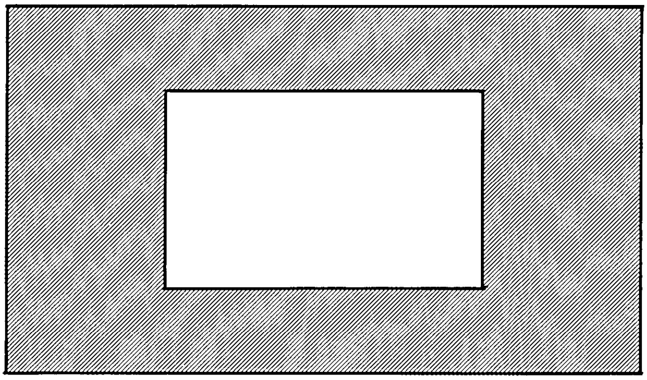

Figure 4-1: Text Windows

Screen coordinates 1,1 (upper left corner) is now the upper left corner of the *window*, not of the physical screen. The screen outside the window is simply not available, and the window behaves as it were the entire screen. You may insert, delete, and scroll lines, and lines will wrap around if too long.

4.3.1 Graphics Windows

In any of the graphics modes, the *GraphWindow* procedure allows you to define any area on the screen as the active window:

```
GraphWindow(X1,Y1,X2,Y2);
```

The graphics windows behave exactly as the text windows described above.

The default graphics window is `GraphWindow(0,0,319,199);` in 320X200-dot modes and `GraphWindow(0,0,639,199);` in 640X200-dot mode, i.e. the entire screen.

Graphics is displayed with clipping, i.e. you may *Draw* between two coordinates outside the window, and only the part of the line that falls within the window will be shown:

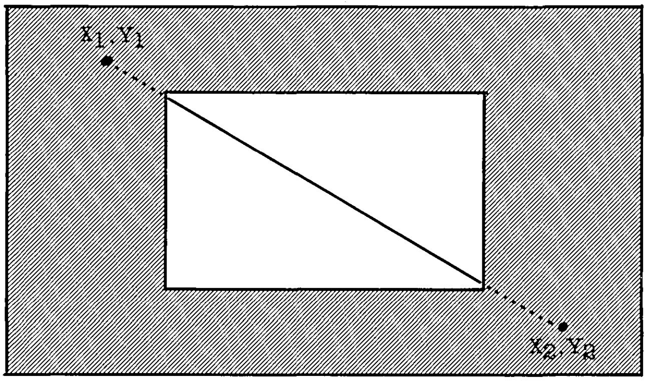

Figure 4-2: Graphics Windows

*Plot*ting outside the window is ignored.

4.4 Sound

The PC's speaker is accessed through the standard procedure *Sound*:

```
Sound(I);
```

where *I* is an integer expression specifying the frequency in Hertz. The specified frequency will be emitted until the speaker is turned off with a call to the *NoSound* standard procedure:

```
NoSound
```

The following example program will emit a 440-Hertz beep for half a second:

```
begin
  Sound(440);
  Delay(500);
  NoSound;
end.
```

Notes:

5. SUMMARY OF VERSION 2 ADDITIONS

reserved word
overlay

procedure
Dispose(**var** *P*: Pointer);
Draw(X_1,Y_1,X_2,Y_2,Color);
FreeMem(**var** *P*: Pointer, *I*: Integer);
GraphBackground(*Color*:Integer);
GraphColorMode;
GraphMode;
GraphWindow(X_1,Y_1,X_2,Y_2,*Color*:Integer);
HiRes;
HiResColor(*Color*:Integer);
NoSound;
Palette(*Color*:Integer);
Plot(*X*,*Y*,*Color*:Integer);
Sound(*I*: Integer);
TextBackground(*Color*:Integer);
TextColor(*Color*:Integer);
TextMode(*Color*:Integer);
Window(X_1,Y_1,X_2,Y_2 ,*Color*:Integer);

function
MaxAvail:Integer;
WhereX:Integer;
WhereY:Integer;

pre-defined constants

BW40:Integer;	= 0
C40:Integer;	= 1
BW80:Integer;	= 2
C80:Integer;	= 3
Black:Integer;	= 0
Blue:Integer;	= 1
Green:Integer;	= 2
Cyan:Integer;	= 3
Red:Integer;	= 4
Magenta:Integer;	= 5
Brown:Integer;	= 6
LightGray:Integer;	= 7
DarkGray:Integer;	= 8
LightBlue:Integer;	= 9
LightGreen:Integer;	= 10
LightCyan:Integer;	= 11
LightRed:Integer;	= 12
LightMagenta:Integer;	= 13
Yellow:Integer;	= 14
White:Integer;	= 15
Blink:Integer;	= 16

6. TURBO-87 PASCAL

TURBO-87 is a special version of TURBO Pascal which uses the Intel 8087 math-processor for real number arithmetic, providing a significant gain in speed and precision. TURBO-87 will compile and run any program written for standard TURBO Pascal; the only difference being in real number processing and real number format.

The TURBO-87 package includes the standard TURBO Pascal compiler. You are thus free to produce your programs in either format. TURBO-87 programs will **not** run on a computer without the 8087-chip installed, whereas the opposite will work.

Files On the Distribution Diskette

In addition to the files listed in section 1.4 of the Reference Manual, the distribution diskette contains the file

TURBO-87.COM

(TURBO-87.CMD in CP/M-86). This file contains the special TURBO-87 compiler. If you need to install it with TINST, you must temporarily rename it to TURBO.COM (or .CMD).

Internal Data Format

The 8087 chip supports a range of data types. The one used by TURBO-87 is the *long real* which is a 64-bit real yielding 16 digits accuracy and a range of 4.19E-307 to 1.67E+308.

This 8-byte real is, of course, not compatible with standard TURBO Pascal's 6-byte real. This, however, should only be a problem if you develop programs in both standard and 87 versions which must interchange data. The trick then is simply to provide an interchange-format between the programs in which you transfer reals e.g. on ASCII format.

A. SUBJECT INDEX

Version 2.0 Supplement

USER's COPY

Program License agreement

On the condition that you sign and return this license agreement to Borland International Inc., you are granted a non exclusive and non transferable license to use TURBO Pascal on one CPU only. Failure to sign this agreement and still use the software is illegal.

The license includes the right to develop programs with TURBO Pascal for the purpose of re-sale. Although the TURBO run-time library remains the property of Borland Inc., and is copyrighted by Borland Inc., no royalty shall be payable to Borland Inc. for programs developed with the TURBO Pascal system.

TURBO Pascal remains the property of Borland International Inc. and giving away or selling copies of TURBO Pascal is theft of Borland's property and will be prosecuted to the fullest extent of the law by the lawyers of Borland International.

The Reference Manual is, of course, your property but copyrighted by Borland International Inc.

Borland warrants that all material furnished by Borland constitutes an accurate original manufacture and will replace any such material proven to be defective, provided that such defect is found and reported *within ten days after purchase*. Borland makes no other express or implied warranties with regard to performance or accuracy of TURBO Pascal and pertaining documentation and specifically disclaims any implied warranties of fitness for any particular purpose. Borland shall not be held responsible for any consequential damages that you may possibly incur through the use of TURBO Pascal, whether through Borland's negligence or not.

Termination of License. Any breach of one or more of the provisions of this agreement shall constitute an immediate termination of this agreement. Nevertheless, I agree that in the event of such termination, all provisions of this agreement which protect the rights of Borland shall remain in force.

I hereby acknowledge that I have read this agreement, understand it and agree to be bound by its terms and conditions.

Name and signature: ____

Address: ____

____________________________ My serial no: ____________________________