Borland Style Graphics
for Dev C++

Mr. Dave Clausen
La Cañada High School
The Text Screen

- The text screen contains 25 lines with a capacity of holding 80 columns of textual characters.

- $80 \times 25 = 2,000$ positions

- But there are actually over 2,000 positions on a display screen.

- The screen consists of pixels (picture elements) that it uses to represent the textual characters and symbols.
Graphics Setup

Here are the steps that you need to follow to use “Borland Style Graphics” source code in Dev C++:

1. Tell the compiler that graphics commands will be used.
2. Initialize the Graphics Screen
3. Close the graphics screen after you have finished drawing your graphics.
Graphics Setup 2

1) To tell the compiler that graphics commands will be used, include the preprocessor directive:
#include <graphics.h>
Graphics Setup 3

• 2) To initialize the graphics screen
  
  `initwindow(640,480);`

After you are finished drawing, you need to use the `while(!kbhit());` command to leave the picture on the screen, or use `cin.get();`

The last choice requires: `#include <iostream.h>`

• 5) Then close the graphics screen, using:
  
  `closegraph();`
Fundamentals of Graphics

- The Graphics Screen.
- Color Options.
- Graphics Mode.
- Drawing Lines
- Line Style
- Clearing the Screen.
- Plotting Points.
The Graphics Screen

- If you have a VGA graphics card or better in your computer, then the graphics screen has 640 pixels across and 480 pixels down.

- $640 \times 480 = 307,200$ pixels

- The upper left corner is position $(0, 0)$

- The lower right corner is position $(639, 479)$
  - Remember, the computer starts counting with zero.
The Graphics Screen Dimensions

(0, 0)  (639, 0)

(0, 479)  (639, 479)
Background Color Options

- You can select the color of the background.
- This is done before drawing anything in the foreground (otherwise your drawing will disappear.)
- To select the background color use the command.
  - `setbkcolor(number);`
    - Where (number) is a numeric constant from 0 through 15, or the symbolic constant that represents the color.
Color Options

- You select a foreground or “drawing” color by using the following command:

```
setcolor(number);
```

- Where (number) is a numeric constant from 0 through 15, or the symbolic constant that represents the color.
Color Names

Here are the color numbers and names:

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
The Current Pointer.

The current pointer is an invisible pointer that keeps track of the current pixel position. It is the equivalent of the visible cursor in text mode.
To move the pointer to a location on the graph without drawing anything, use the command:

```
moveto (X,Y);
```

- This is like `PenUp (PU)` in LOGO.

To draw lines from the current pointer’s position to another point on the graph, use the command:

```
lineto (X,Y);
```

- This is like `PenDown (PD)` in LOGO or `SetXY (x, y)`.
Graphics Figures

- Lines
- Rectangles
- Circles
- Arcs
- Ellipses
- Points
Instead of using the commands: moveto and lineto, we can draw a line using one command:

\[
\text{line}(x_1, y_1, x_2, y_2);
\]

The points \((x_1, y_1)\) describe the beginning of the line, while \((x_2, y_2)\) describes the endpoint of the line.

The numbers \(x_1, y_1, x_2, y_2\) are integers.
Rectangles

Rectangles can be drawn in different ways using lineto, moveto, moverel, and linerel. But an easier and faster way is using the Rectangle procedure which draws a rectangle in the default color and line style with the upper left at X1, Y1 and lower right X2, Y2.

```
rectangle (x1, y1, x2, y2);
```
Circles

Circles can be drawn using the circle procedure. This draws a circle in the default color and line style with center at X, Y, radius in the X direction of Xradius, and corresponding Y radius.

circle (x, y, radius);
Arcs

This procedure draws a circular arc in the default color and line style based upon a circle with center X, Y and given X radius. The arc begins at an angle of StartAngle and follows the circle to EndAngle. The angles are measured in degrees from 0 to 360 counter-clockwise where 0 degrees is directly right.

arc ( x, y, startangle, endangle, radius);
Visualizing Arcs
Starting & Ending Angles

Starting Angle
Ending Angle

0 90
180 270

Mr. Dave Clausen
Ellipses

Draws an elliptical arc in the default color and line style based upon an ellipse with center X, Y and given radii.
The arc begins at an angle to Start Angle and follows the ellipse to End Angle. The angles are measured in degrees from 0 to 360 counterclockwise where 0 degrees is directly right.

ellipse ( x, y, startangle , endangle, x_radius, y_radius);
Plotting Points

- The Maximum value for X can be found using:
  getmaxx()

- The Maximum value for Y can be found using:
  getmaxy()

- To Plot a point:
  putpixel ( x_value, y_value, color);
  For example: putpixel (100, 100, WHITE);
Sample Program

- Let’s look at a program with a line, rectangle, circle, arc, ellipse, and a point.

Objects.cpp
Line Style

Setting the line style.

All lines have a default line mode, but Turbo C++ allows the user to specify three characteristics of a line: style, pattern, and thickness.

Use the command:

setlinestyle (style, pattern, thickness);
Here are the names of the line styles and thickness:

<table>
<thead>
<tr>
<th>Line Style</th>
<th>Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID_LINE</td>
<td>NORM_WIDTH</td>
</tr>
<tr>
<td>DOTTED_LINE</td>
<td></td>
</tr>
<tr>
<td>CENTER_LINE</td>
<td>THICK_WIDTH</td>
</tr>
<tr>
<td>DASHED_LINE</td>
<td></td>
</tr>
<tr>
<td>USERBIT_LINE</td>
<td></td>
</tr>
</tbody>
</table>
Line Style Patterns

The names of the line patterns are:

SOLID_LINE = 0
DOTTED_LINE = 1
CENTER_LINE = 2
DASHED_LINE = 3
Filling Patterns

• Selecting Pattern and Color
• Filling Regions
• Getting a Pixel
Selecting Pattern and Color

Use the command SetFillStyle for setting the pattern and color for the object that you wish to fill.

\[ \text{setfillstyle ( pattern, color); } \]
# Pattern Names

Here are the names of available patterns:

<table>
<thead>
<tr>
<th>Values</th>
<th>Causing filling with</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY_FILL</td>
<td>Background Color</td>
</tr>
<tr>
<td>SOLID_FILL</td>
<td>Solid Color</td>
</tr>
<tr>
<td>LINE_FILL</td>
<td>Horizontal Lines</td>
</tr>
<tr>
<td>LTSLASH_FILL</td>
<td>Thin diagonal lines</td>
</tr>
<tr>
<td>SLASH_FILL</td>
<td>Thick diagonal lines</td>
</tr>
<tr>
<td>BKSLASH_FILL</td>
<td>Thick diagonal backslashes</td>
</tr>
<tr>
<td>LTBKSLASH_FILL</td>
<td>Light backslashes</td>
</tr>
<tr>
<td>HATCH_FILL</td>
<td>Thin cross hatching</td>
</tr>
<tr>
<td>XHATCH_FILL</td>
<td>Thick cross hatching</td>
</tr>
<tr>
<td>INTERLEAVE_FILL</td>
<td>Interleaving lines</td>
</tr>
<tr>
<td>WIDE_DOT_FILL</td>
<td>Widely spaced dots</td>
</tr>
<tr>
<td>CLOSE_DOT_FILL</td>
<td>Closely spaced dots</td>
</tr>
</tbody>
</table>
Filling Regions

- After selecting a color and pattern, floodfill is used to fill the desired area.
- `floodfill ( x, y, border_color );`
- This “paints out” the desired color until it reaches border color.
- **Note:** The border color must be the same color as the color used to draw the shape.
- Also, you can only fill completely “closed” shapes.

Program10_4.cpp
Filling “Special” Regions

- To draw a filled ellipse:
  \[\text{fillellipse (xcoordinate, ycoordinate, xradius, yradius);}\]

- To draw a filled rectangle:
  \[\text{bar (x1, y1, x2, y2);}\]

- To draw a filled 3D rectangle:
  \[\text{bar3d(x1, y1, x2, y2, depth, topflag); //depth is width of the 3D rectangle, if topflag is non-0 a top is added to the bar}\]

- To draw a filled section of a circle:
  \[\text{pieslice (x, y, startangle, endangle, xradius);}\]
Text Output on the Graphics Screen

To write a literal expression on the graphics screen using the location specified by (x, y) use the command:

`outtextxy(x, y, "literal expression");`

`outtextxy(x, y, string_variable);`

Note: These are not “apstring” type strings. They are C++ standard Strings.
Text Styles

To set the values for the text characteristics, use:
settextstyle ( font, direction, charsize);

**Font**

- **DEFAULT_FONT**
  - Direction: HORIZ_DIR = Left to right

- **TRIPLEX_FONT**
  - Direction: VERT_DIR = Bottom to top

- **SMALL_FONT**

- **SANS_SERIF_FONT**

- **GOTHIC_FONT**

- **SCRIPT_FONT**

- **SIMPLEX_FONT**

- **TRIPLEX_SCR_FONT**

**Fonts continued**

- **COMPLEX_FONT**

- **EUROPEAN_FONT**

- **BOLD_FONT**
Text Styles
Font Sizes

CharSize

1 = Default (normal)
2 = Double Size
3 = Triple Size
4 = 4 Times the normal
5 = 5 Times the normal
.....
10 = 10 Times the normal
Text Justification

To set the way that text is located around the point specified use the command:

```
settextjustification (horizontal,vertical);
```

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_TEXT</td>
<td>TOP_TEXT</td>
</tr>
<tr>
<td>CENTER_TEXT</td>
<td>BOTTOM_TEXT</td>
</tr>
<tr>
<td>RIGHT_TEXT</td>
<td></td>
</tr>
</tbody>
</table>

Program10_2.cpp
Clearing the Screen

- Here is the way to clear the graphics screen.
- When in graphics mode use:
  
cleardevice();  //include <graphics.h>
Text

Height & Width

- Returns the height, in pixels, of string S if it were to be written on the graphics screen using the current defaults.
  \text{textheight} (S \text{ string});

- Returns the width, in pixels, of string S if it were to be written on the graphics screen using the current defaults.
  \text{textwidth} (S \text{ string});
Getting a Pixel

To return the color number corresponding to the color located at the point: X, Y use the command:

`getpixel (x, y);`
Useful Non Graphic Commands

- **kbhit()**
  - checks to see if a keystroke is currently available
  - If a keystroke is available, returns a nonzero integer.
  - If a keystroke is not available, returns a zero.

- Any available keystrokes can be retrieved with `getch()`.