NINE WEEK

Graphics

getmaxx and getmaxy <GRAPHICS.H>

Declaration:

- int far getmaxx(void);
- int far getmaxy(void);

Remarks:

- getmaxx returns the maximum x value (screen-relative) for the current graphics driver and mode.
- getmaxy returns the maximum y value (screen-relative) for the current graphics driver and mode.

getx, gety <GRAPHICS.H>

Declaration:

- int far getx(void);
- int far gety(void);

Remarks:

- getx returns the x-coordinate of the current graphics position.
- gety returns the y-coordinate of the current graphics position.

moverel, moveto <GRAPHICS.H>

Declaration:

- void far moverel(int dx, int dy);
- void far moveto(int x, int y);

Remarks:

- moverel moves the current position (CP) dx pixels in the x direction and dy pixels in the y direction.

- moveto moves the current position (CP) to viewport position (x, y).

getpixel, putpixel <GRAPHICS.H>

Declaration:

- unsigned far getpixel(int x, int y);
- void far putpixel(int x, int y, int color);

Remarks:

- getpixel gets the color of the pixel located at (x,y).
- putpixel plots a point in the color defined by color at (x,y).

line, linerel, lineto <GRAPHICS.H>

Declaration:

- void far line(int x1, int y1, int x2, int y2);
- void far linerel(int dx, int dy);
- void far lineto(int x, int y);

Remarks:

- line draws a line from (x1, y1) to (x2, y2) using the current color, line style, and thickness. It does not update the current position (CP).

- linerel draws a line from the CP to a point that is a relative distance (dx, dy) from the CP, then advances the CP by (dx, dy).

- lineto draws a line from the CP to (x, y), then moves the CP to (x, y).

arc, circle, pieslice <GRAPHICS.H>

Declaration:

- void far arc(int x, int y, int stangle, int endangle, int radius);
- void far circle(int x, int y, int radius);
- void far pieslice(int x, int y, int stangle, int endangle, int radius);

Remarks:

arc draws a circular arc in the current drawing color.

circle draws a circle in the current drawing color.

pieslice draws a pie slice in the current drawing color, then fills it using the current fill pattern and fill color.

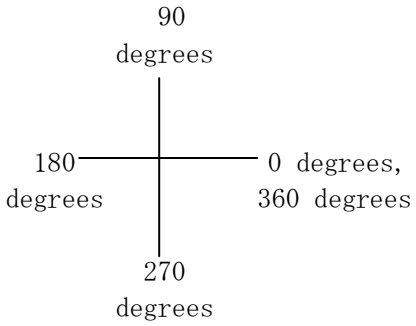
Argument	What It Is/Does
(x,y)	Center point of arc, circlew, or pie slice
stangle	Start angle in degrees
endangle	End angle in degrees
radius	Radius of arc, circle, and pieslice

The arc or slice travels from stangle to endangle.

If stangle = 0 and endangle = 360, the call to arc draws a complete circle.

If your circles are not perfectly round, use set aspect ratio to adjust the aspect ratio.

Angle for arc, circle, and pieslice (counter-clockwise)



The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

rectangle <GRAPHICS.H>

Declaration:

void far rectangle(int left, int top, int right, int bottom);

Remarks:

rectangle draws a rectangle in the current line style, thickness, and drawing color.

(left,top) is the upper left corner of the rectangle, and (right,bottom) is its lower right corner.

ellipse, fillellipse, sector <GRAPHICS.H>

Declaration:

- void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);
- void far fillellipse(int x, int y,int xradius, int yradius);
- void far sector(int x, int y, int stangle, int endangle,int xradius, int yradius);

Remarks:

ellipse draws an elliptical arc in the current drawing color.

fillellipse draws an ellipse, then fills the ellipse with the current fill color and fill pattern.

sector draws and fills an elliptical pie slice in the current drawing color, then fills it using the pattern and color defined by setfillstyle or setfillpattern.

Argument : What It Is

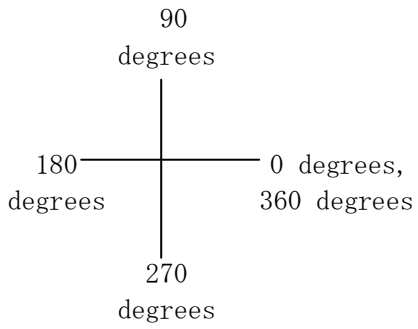
(x,y) : Center of ellipse
xradius : Horizontal axis
yradius : Vertical axis
stangle : Starting angle

endangle : Ending angle

The ellipse or sector travels from stangle to endangle.

If stangle = 0 and endangle = 360, the call to ellipse draws a complete ellipse.

Angle for ellipse, fillellipse, and sector (counter-clockwise)



The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

bar <GRAPHICS.H>

Declaration: void far bar(int left, int top, int right, int bottom);
Remarks:
bar draws a filled-in, rectangular, two-dimensional bar.
The bar is filled using the current fill pattern and fill color. bar does not outline the bar.
To draw an outlined two-dimensional bar, use bar3d with depth = 0.

Parameters : What they are
(left, top) : the rectangle's upper left corner
(right, bottom) : the rectangle's lower right corner
The coordinates are in pixels.

getcolor, setcolor <GRAPHICS.H>

Declaration:
- int far getcolor(void);
- void far setcolor(int color);

Remarks:
getcolor returns the current drawing color.
setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.
To select a drawing color with setcolor, you can pass either the color number or the equivalent color name.
The drawing color is the value that pixels are set to when the program draws lines, etc.

COLORS (text mode)

Constant	Value	Back-grnd?	Fore-grnd

BLACK	: 0	: Yes	: Yes
BLUE	: 1	: Yes	: Yes
GREEN	: 2	: Yes	: Yes
CYAN	: 3	: Yes	: Yes
RED	: 4	: Yes	: Yes
MAGENTA	: 5	: Yes	: Yes
BROWN	: 6	: Yes	: Yes
LIGHTGRAY	: 7	: Yes	: Yes
DARKGRAY	: 8	: No	: Yes
LIGHTBLUE	: 9	: No	: Yes
LIGHTGREEN	: 10	: No	: Yes
LIGHTCYAN	: 11	: No	:Yes
LIGHTRED	: 12	: No	:Yes
LIGHTMAGENTA	: 13	: No	: Yes
YELLOW	: 14	: No	: Yes
WHITE	: 15	: No	: Yes
BLINK	:128	: No	: ***

*** To display blinking characters in text mode, add BLINK to the foreground color. (Defined in CONIO.H.)

getmaxcolor <GRAPHICS.H>

Returns maximum color value

Declaration: int far getmaxcolor(void);

Remarks:
getmaxcolor returns the highest valid color value that can be passed to setcolor for the current graphics driver and mode.

getbkcolor, setbkcolor <GRAPHICS.H>

Declaration:

```
int far getbkcolor(void);
void far setbkcolor(int color);
```

Remarks:
getbkcolor returns the current background color.

setbkcolor sets the background to the color specified by color.

cleardevice <GRAPHICS.H>

Clears the graphics screen

Declaration: void far cleardevice(void);

Remarks:
cleardevice erases the entire graphics screen and moves the CP (current
position) to home (0,0).

(Erasing consists of filling with the current background color.)

setlinestyle <GRAPHICS.H>

Sets the current line style and width or pattern
Declaration:
  void far setlinestyle(int linestyle, unsigned upattern, int thickness);

Remarks:
setlinestyle sets the style for all lines drawn by line, lineto, rectangle, drawpoly, etc.

line_styles <GRAPHICS.H>

Enum: Line styles for getlinesettings and setlinestyle.

Name	Value	Meaning
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

line_widths <GRAPHICS.H>

Enum: Line widths for getlinesettings and setlinestyle.

Name	Value	Meaning
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

setfillstyle <GRAPHICS.H>

Declaration: void far setfillstyle(int pattern, int color);

Remarks:
setfillstyle sets the current fill pattern and fill color.

To set a user-defined fill pattern, do not give a pattern of 12 (USER_FILL)to setfillstyle; instead, call setfillpattern.

The enumeration fill_patterns, defined in GRAPHICS.H, gives names for the predefined fill patterns, plus an indicator for a user-defined pattern.

fill_patterns <GRAPHICS.H>

Enum: Fill patterns for getfillsettings and setfillstyle.

Names	Value	Means Fill With...
EMPTY_FILL	0	Background color
SOLID_FILL	1	Solid fill
LINE_FILL	2	---
LTSLASH_FILL	3	///

```
SLASH_FILL      :  4  : ///, thick lines
BKSLASH_FILL    :  5  : \\\, thick lines
LTBKSLASH_FILL  :  6  : \\\
HATCH_FILL      :  7  : Light hatch
XHATCH_FILL     :  8  : Heavy crosshatch
INTERLEAVE_FILL :  9  : Interleaving lines
WIDE_DOT_FILL   : 10  : Widely spaced dots
CLOSE_DOT_FILL  : 11  : Closely spaced dots
USER_FILL       : 12  : User-defined fill pattern
```

All but EMPTY_FILL fill with the current fill color. EMPTY_FILL uses the current background color.

floodfill <GRAPHICS.H>

Declaration: void far floodfill(int x, int y, int border);

Remarks:

floodfill fills an enclosed area on bitmap devices.

The area bounded by the color border is flooded with the current fill pattern and fill color.

(x,y) is a "seed point".

- If the seed is within an enclosed area, the inside will be filled.
- If the seed is outside the enclosed area, the exterior will be filled.

Use fillpoly instead of floodfill whenever possible so you can maintain code compatibility with future versions.

Example:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* initialize graphics, local variables
*/
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
      /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();
```

```
    /* select drawing color */
    setcolor(getmaxcolor());

    /* select fill color */
    setfillstyle(SOLID_FILL, getmaxcolor());

    /* draw a border around the screen */
    rectangle(0, 0, maxx, maxy);

    /* draw some circles */
    circle(maxx / 3, maxy /2, 50);
    circle(maxx / 2, 20, 100);
    circle(maxx-20, maxy-50, 75);
    circle(20, maxy-20, 25);

    /* wait for a key */
    getch();

    /* fill in bounded region */
    floodfill(2, 2, getmaxcolor());

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

textheight, textwidth <GRAPHICS.H>

Declaration:

- int far textheight(char far *textstring);
- int far textwidth(char far *textstring);

Remarks:

- textheight takes the current font size and multiplication factor, and determines the height of textstring in pixels.

- textwidth takes the string length, current font size, and multiplication factor, and determines the width of textstring in pixels.

settextstyle <GRAPHICS.H>

Declaration:

void far settextstyle(int font, int direction, int charsize);

Remarks:

settextstyle sets the text font, the direction in which text is displayed, and the size of the characters.

A call to settextstyle affects all text output by outtext and outtextxy.

font

One 8x8 bit-mapped font and several "stroked" fonts are available. The 8x8 bit-mapped font, the default, is built into the graphics system.

The enumeration font_names, defined in GRAPHICS.H, provides names for the different font settings.

font_names <GRAPHICS.H>

Enum: Names for BGI fonts

Name	Value	Meaning
DEFAULT_FONT	0	8x8 bit-mapped font
TRIPLEX_FONT	1	Stroked triplex font
SMALL_FONT	2	Stroked small font
SANS_SERIF_FONT	3	Stroked sans-serif font
GOTHIC_FONT	4	Stroked gothic font

direction

Font directions supported are horizontal text (left to right) and vertical text (rotated 90 degrees counterclockwise).

The default direction is HORIZ_DIR.

Name	Value	Direction
HORIZ_DIR	0	Left to right
VERT_DIR	1	Bottom to top

charsize

The size of each character can be magnified using the charsize factor. If charsize is non-zero, it can affect bit-mapped or stroked characters.

outtext, outtextxy <GRAPHICS.H>

Declaration:

- void far outtext(char far *textstring);
- void far outtextxy(int x, int y, char far *textstring);

Remarks:

outtext and outtextxy display a text string, using the current justification settings and the current font, direction, and size.

- outtext outputs textstring at the current position (CP)
- outtextxy displays textstring in the viewport at the position (x, y)

To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string.
If a string is printed with the default font using outtext or outtextxy, any part of the string that extends outside the current viewport is truncated.
With outtext, if the horizontal text justification is LEFT_TEXT and the text direction is HORIZ_DIR, the CP's x-coordinate is advanced by textwidth(textstring).
Otherwise, the CP remains unchanged.
outtext and outtextxy are for use in graphics mode; they will not work in text mode.

outtext example

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
```

```
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   /* move the C.P. to the center of the screen */
   moveto(midx, midy);

   /* output text starting at the C.P. */
   outtext("This ");
   outtext("is ");
   outtext("a ");
   outtext("test.");
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getimage, putimage <GRAPHICS.H>

- getimage saves a bit image of the specified region into memory
- putimage outputs a bit image onto the screen

Declaration:

- void far getimage(int left, int top, int right, int bottom,void far *bitmap);
- void far putimage(int left, int top, void far *bitmap, int op);

Remarks:

- getimage copies an image from the screen to memory.

- putimage puts the bit image previously saved with getimage back onto the screen, with the upper left corner of the image placed at (left,top).

Argument	What It Is/Does
bitmap	Points to the area in memory where the bit image is stored. The first two words of this area are used for the width and height of the rectangle. The remainder holds the Image itself.
bottom left right top	(left, top) and (right, bottom) define the rectangular screen area from which getimage copies the bit image. (left, top) is where putimage places the upper left corner of the stored image.
op	Specifies a combination operator that controls how the color for each destination pixel onscreen is computed, based on the pixel already onscreen and the corresponding source pixel in memory.

The enumeration putimage_ops, defined in GRAPHICS.H, gives names to the putimage combination operators.

putimage_ops <GRAPHICS.H>

Enum: Operators for putimage

Constant:	Value	: Meaning
COPY_PUT :	0	: Copies source bitmap onto screen
XOR_PUT :	1	: Exclusive ORs source image with that already onscreen
OR_PUT :	2	: Inclusive ORs image with that already onscreen
AND_PUT :	3	: ANDs image with that already onscreen
NOT_PUT :	4	:Copy the inverse of the source

imagesize <GRAPHICS.H>

Declaration:

unsigned far imagesize(int left, int top, int right, int bottom);

Remarks:

imagesize determines the size of the memory area required to store a bit image.

Return Value:

- On success, returns the size of the required memory area in bytes.
- On error (if the size required for the selected image is >= (64K - 1) bytes),
  returns 0xFFFF (-1)

getimage example

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

void save_screen(void far *buf[4]);
void restore_screen(void far *buf[4]);

int maxx, maxy;

int main(void)
{
   int gdriver=DETECT, gmode, errorcode;
   void far *ptr[4];

   /* auto-detect the graphics driver and mode */
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult(); /* check for any errors */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
   maxx = 50; //getmaxx();
   maxy = 60;  //getmaxy();

   /* draw an image on the screen */
   rectangle(0, 0, maxx, maxy);
   line(0, 0, maxx, maxy);
   line(0, maxy, maxx, 0);
   save_screen(ptr);    /* save the current screen */
   getch();             /* pause screen */
   cleardevice();       /* clear screen */
   getch();
   restore_screen(ptr); /* restore the screen */
   getch();             /* pause screen */
```

```
   closegraph();
   return 0;
}
void save_screen(void far *buf[4])
{
   unsigned size;
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;
   size = imagesize(0, ystart, maxx, yend);
/* get byte size of image */

   for (block=0; block<=3; block++)
   {
      if ((buf[block] = farmalloc(size)) == NULL)
      {
         closegraph();
         printf("Error: not enough heap space in save_screen().\n");
         exit(1);
      }
      getimage(0, ystart, maxx, yend, buf[block]);
      ystart = yend + 1;       yend += yincr + 1;
   }
}
void restore_screen(void far *buf[4])
{
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;

   for (block=0; block<=3; block++)
   {
      putimage(0, ystart, buf[block], COPY_PUT);
      farfree(buf[block]);
      ystart = yend + 1;       yend += yincr + 1;
   }
}
```

Applications on Graphics

Q) Draw football stadd

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");       getch();
      exit(1);             /* return with error code */
   }

   /* draw green bar with white border */
```

```
/*   setfillstyle(1, 2);
   bar(100, 100, 500, 300);
   setcolor(15);
   rectangle(100, 100, 500, 300);*/
 //or
   setcolor(15);
   rectangle(100, 100, 500, 300);
   setfillstyle(1, 2);
   floodfill(200,200,15);

   line(300,100,300,300);
   circle(300,200,30);
   rectangle(100,150,160,250);
   rectangle(100,170,130,230);
   rectangle(500,150,440,250);
   rectangle(500,170,470,230);
   arc(160,200,270,90,20);
   arc(440,200,90,270,20);

   arc(100,100,270,360,10);
   arc(500,100,180,270,10);
   arc(100,300,0,90,10);
   arc(500,300,90,180,10);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

Q) Draw Tanis stadd

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);             /* return with error code */
   }

   /* draw green bar with white border */
/*   setfillstyle(1, 2);
```

```
    bar(100, 100, 400, 300);
    setcolor(15);
    rectangle(100, 100, 400, 300);*/
  //or
    setcolor(15);
    rectangle(100, 100, 400, 300);
    setfillstyle(1, 2);
    floodfill(200,200,15);

    rectangle(100, 150, 400, 250);
    rectangle(175, 150, 325, 250);

    setfillstyle(1, 15);
    circle(500,200,10);
    floodfill(500,200,15);
    bar(452,180,458,250);

    setfillstyle(8, 14);
    ellipse(455,160,0,360,10,20);
    floodfill(455,160,15);


    /* clean up */
    getch();
    closegraph();
    return 0;
}
```