COMPUTER GRAPHICS LAB PRACTICAL

Declaration :- void arc(int x, int y, int stangle, int endangle, int radius);
arc function is used to draw an arc with center (x,y) and stangle specifies starting angle, endangle specifies the end angle and last parameter specifies the radius of the arc. arc function can also be used to draw a circle but for that starting angle and end angle should be 0 and 360 respectively**.**

Arc draw program

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

main()
{
   int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\TC\\BGI");

arc(100, 100, 0, 135, 50);

getch();
 closegraph();
return 0;
}
```

Bar function in c

Declaration: - void bar(int left, int top, int right, int bottom);

Bar function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner are required to draw the bar. Left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner. Current fill pattern and fill color is used to fill the bar. To change fill pattern and fill color use setfillstyle.

Code for Bar draw

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

        main()
        {
          int gd = DETECT, gm;

        initgraph(&gd, &gm, "C:\\TC\\BGI");

        bar(100, 100, 200, 200);

         getch();
         closegraph();
        return 0;
}
```

Draw 3d Bar

Declaration: - void bar3d (int left, int top, int right, int bottom, int depth, int topflag);

Bar3d function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner of bar are required to draw the bar. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner, depth specifies the depth of bar in pixels, topflag determines whether a 3 dimensional top is put on the bar or not (if it is non-zero then it is put otherwise not). Current fill pattern and fill color is used to fill the bar. To change fill pattern and fill color use setfillstyle

Code for bar3d

```
include <graphics.h.
include <conio.h>

main()
{
int gd = DETECT, gm;

 initgraph(&gd, &gm, "C:\\TC\\BGI");

   bar3d(100, 100, 200, 200, 20, 1);

getch();
closegraph();
return 0;
}
```

Draw Circle Declaration :-void circle (int x, int y, int radius);

circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle. The code given below draws a circle.

Code for circle

```
#include<graphics.h>

#include<conio.h>

  main()
  {
   int gd = DETECT, gm;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   circle(100, 100, 50);

  getch();
  closegraph();
   return 0;
  }
```

Ellipse function in c

Declarations of ellipse function :-
void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);
Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.

Code for ellipse

```
        #include<graphics.h>
        #include<conio.h>
          main()
          {
             int gd = DETECT, gm;

             initgraph(&gd, &gm, "C:\\TC\\BGI");

             ellipse(100, 100, 0, 360, 50, 25);

            getch();
             closegraph();
            return 0;
         }
```

getcolor function

getcolor function returns the current drawing color.

Declaration:- int getcolor();
e.g. a = getcolor(); // a is an integer variable
if current drawing color is WHITE then a will be 15.
See colors in c graphics
Code for getcolor:

```
main()
{
   int gd = DETECT, gm, drawing_color;
   char a[100];

   initgraph(&gd,&gm,"C:\\TC\\BGI");

   drawing_color = getcolor();

   sprintf(a,"Current drawing color = %d", drawing_color);
   outtextxy( 10, 10, a );

   getch();
   closegraph();
   return 0;
}
```

getbkcolor function in c

getbkcolor function returns the current background color

Declaration : int getbkcolor();

e.g. color = getbkcolor(); // color is an int variable
if current background color is GREEN then color will be 2.

Code for getbkcolor :-

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

main()
{
   int gd = DETECT, gm, bkcolor;
   char a[100];

   initgraph(&gd,&gm,"C:\\TC\\BGI");

   bkcolor = getbkcolor();

   sprintf(a,"Current background color = %d", bkcolor);
   outtextxy( 10, 10, a);

   getch();
   closegraph();
   return 0;
}
```

getmaxcolor function

getmaxcolor function returns maximum color value for current graphics mode and driver. Total number of colors available for current graphics mode and driver are (getmaxcolor() + 1) as color numbering starts from zero.

Declaration: - int getmaxcolor ();

Code for getmaxcolor :-

```
#include <graphics.h>

#include<conio.h>

main()
{
   int gd = DETECT, gm, max_colors;
   char a[100];

   initgraph(&gd,&gm,"C:\\TC\\BGI");

   max_colors = getmaxcolor();

        sprintf(a,"Maximum number of colors for current graphics mode and driver =
%d",max_colors+1);
   outtextxy(0, 40, a);

   getch();

   closegraph();

   return 0;

}
```

Graphics program to draw circles in circles

This program draws circles in circles in two different colors.

Code for draw circle in circle:-

```
#include <graphics.h>

#include<conio.h.>

#include<dos.h>


  main()
  {
   int gd = DETECT, gm, x, y, color, angle = 0;

   struct arccoordstype a, b;

  initgraph(&gd, &gm, "C:\\TC\\BGI");

  delay(2000);

  while(angle<=360)
   {
          setcolor(BLACK);

          arc(getmaxx()/2,getmaxy()/2,angle,angle+2,100);

          setcolor(RED);

          getarccoords(&a);

          circle(a.xstart,a.ystart,25);

          setcolor(BLACK);

          arc(getmaxx()/2,getmaxy()/2,angle,angle+2,150);

          getarccoords(&a);

          setcolor(GREEN);

          circle(a.xstart,a.ystart,25);

          angle = angle+5;

           delay(50);
   }

          getch();

          closegraph();

          return 0;
  }
```

C graphics examples

1. Drawing concentric circles

```
#include <graphics.h>

int main()
{
   int gd = DETECT, gm;
   int x = 320, y = 240, radius;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   for ( radius = 25; radius <= 125 ; radius = radius + 20)
      circle(x, y, radius);

   getch();
   closegraph();
   return 0;
}
```

2. C graphics program moving car

```
#include <graphics.h>
#include <dos.h>

int main()
{
   int i, j = 0, gd = DETECT, gm;

   initgraph(&gd,&gm,"C:\\TC\\BGI");

   settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
   outtextxy(25,240,"Press any key to view the moving car");

   getch();

   for( i = 0 ; i <= 420 ; i = i + 10, j++ )
   {
      rectangle(50+i,275,150+i,400);
      rectangle(150+i,350,200+i,400);
      circle(75+i,410,10);
      circle(175+i,410,10);
      setcolor(j);
      delay(100);

      if( i == 420 )
         break;
      if ( j == 15 )
```

```
            j = 2;

         cleardevice(); // clear screen
      }

   getch();
   closegraph();
   return 0;
}
```

3.

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

main()
{
    int gd = DETECT, gm, height;
    char array[100];

    initgraph(&gd, &gm, "C:\\TC\\BGI");

    height = textheight("C programming");

    sprintf(array,"Textheight = %d",height);
    outtext(array);

    getch();
    closegraph();
    return 0;
}
```

4.

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

main()
{
   int gd = DETECT, gm, width;
   char array[100];

   initgraph(&gd, &gm, "C:\\TC\\BG I");

   width = textwidth("C programming");

   sprintf(array,"Textwidth = %d",width);
   outtext(array);

   getch();
   closegraph();
   return 0;
}
```

setviewport function in c

setviewport function sets the current viewport for graphics output.

Declaration :- void setviewport(int left, int top, int right, int bottom, int clip);

setviewport function is used to restrict drawing to a particular portion on the screen. For example
setviewport(100 , 100, 200, 200, 1);
will restrict our drawing activity inside the rectangle(100,100, 200, 200).
left, top, right, bottom are the coordinates of main diagonal of rectangle in which we wish to restrict our drawing. Also note that the point (left, top) becomes the new origin.

5.

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

main()
{
   int gd = DETECT, gm, midx, midy;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   midx = getmaxx()/2;
   midy = getmaxy()/2;

   setviewport(midx - 50, midy - 50, midx + 50, midy + 50, 1);
   circle(50, 50, 55);

   getch();
   closegraph();
   return 0;
}
```

6.

Settextstyle function is used to change the way in which text appears, using it we can modify the size of text, change direction of text and change the font of text.

Declaration :- void settextstyle(int font, int direction, int charsize);
font argument specifies the font of text, Direction can be HORIZ_DIR (Left to right) or VERT_DIR (Bottom to top).

Different fonts

```
enum font_names
{
   DEFAULT_FONT,
   TRIPLEX_FONT,
   SMALL_FONT,
   SANS_SERIF_FONT,
   GOTHIC_FONT,
   SCRIPT_FONT,
   SIMPLEX_FONT,
   TRIPLEX_SCR_FONT,
   COMPLEX_FONT,
   EUROPEAN_FONT,
   BOLD_FONT
};
```

7.

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

main()
{
   int gd = DETECT, gm, x = 25, y = 25, font = 0;
   initgraph(&gd,&gm,"C:\\TC\\BGI");

   for ( font = 0 ; font <= 10 ; font++)
   {
      settextstyle(font, HORIZ_DIR, 1);
      outtextxy(x, y, "Text with different fonts");
      y = y + 25;
   }

   getch();
   closegraph();
   return 0;
}
```

setlinestyle in c

Declaration:
void setlinestyle(int linestyle, unsigned upattern, int thickness);

Available line styles:

```
enum line_styles
{
   SOLID_LINE,
   DOTTED_LINE,
   CENTER_LINE,
   DASHED_LINE,
   USERBIT_LINE
};
```

8.

```
#include <graphics.h>

main()
{
   int gd = DETECT, gm, c , x = 100, y = 50;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   for ( c = 0 ; c < 5 ; c++ )
   {
       setlinestyle(c, 0, 2);

       line(x, y, x+200, y);
       y = y + 25;
   }

   getch();
   closegraph();
   return 0;
}
```

setfillstyle function in c

setfillstyle function sets the current fill pattern and fill color.

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

Different fill styles:

```
enum fill_styles
{
   EMPTY_FILL,
   SOLID_FILL,
   LINE_FILL,
   LTSLASH_FILL,
   SLASH_FILL,
   BKSLASH_FILL,
   LTBKSLASH_FILL,
   HATCH_FILL,
   XHATCH_FILL,
   INTERLEAVE_FILL,
   WIDE_DOT_FILL,
   CLOSE_DOT_FILL,
   USER_FILL
};
```

C programming source code for setfillstyle

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

main()
{
   int gd = DETECT, gm;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   setfillstyle(XHATCH_FILL, RED);
   circle(100, 100, 50);
   floodfill(100, 100, WHITE);

   getch();
   closegraph();
   return 0;
}
```

setbkcolor function in c

Declaration :- void setbkcolor(int color);

setbkcolor function changes current background color e.g. setbkcolor(YELLLOW) changes the current background color to YELLOW.
Remember that default drawing color is WHITE and background color is BLACK.

C programming code for setbkcolor

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

main()
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");

   outtext("Press any key to change the background color to GREEN.");
   getch();

   setbkcolor(GREEN);

   getch();
   closegraph();
   return 0;
}
```

setcolor function in c

Declaration :- void setcolor(int color);

In Turbo Graphics each color is assigned a number. Total 16 colors are available. Strictly speaking number of available colors depends on current graphics mode and driver.For Example :- BLACK is assigned 0, RED is assigned 4 etc. setcolor function is used to change the current drawing color.e.g. setcolor(RED) or setcolor(4) changes the current drawing color to RED. Remember that default drawing color is WHITE.

C programming code for setcolor

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

main()
{
   int gd = DETECT, gm;
   initgraph(&gd,&gm,"C:\\TC\\BGI");
   circle(100,100,50);           /* drawn in white color */
   setcolor(RED);
   circle(200,200,50);           /* drawn in red color   */

   getch();
   closegraph();
   return 0;
}
```

moverel function in c

moverel function moves the current position to a relative distance.

Declaration :- void moverel(int x, int y);

C programming code for moverel

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

main()
{
   int gd = DETECT, gm, x, y;
   char message[100];

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   moveto(100, 100);
   moverel(100, -100);

   x = getx();
   y = gety();

   sprintf(message, "Current x position = %d and y position = %d", x,
y);
   outtextxy(10, 10, message);

   getch();
   closegraph();
   return 0;
}
```

DDA ALGORITHM

DDA Line (X1, Y1, XN, YN):

Description: Here X1 and Y1 denote the starting x – coordinate and y – coordinate of the line and XN and YN denote the ending x – coordinate and y – coordinate.

```
1. Set M = (YN – Y1) / (XN – X1) [Calculate slope of line]
2. Repeat For I = X1 to XN
3. If (M <= 1) Then
4. Set DX = 1
5. Set DY = M * DX
6. Else
7. Set DY = 1
8. Set DX = DY / M
[End of If]
9. Set X1 = X1 + DX
10. Set Y1 = Y1 + DY
11. Call PutPixel(X1, Y1)
12.     [End of For]
```

Bresenham line algorithm

The **Bresenham line algorithm** is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithms developed in the field of computer graphics. A minor extension to the original algorithm also deals with drawing circles.

While algorithms such as Wu's algorithm are also frequently used in modern computer graphics because they can support antialiasing, the speed and simplicity of Bresenham's line algorithm mean that it is still important. The algorithm is used in hardware such as plotters and in the graphics chips of modern graphics cards. It can also be found in many software graphics libraries. Because the algorithm is very simple, it is often implemented in either the firmware or the hardware of modern graphics cards.

The label "Bresenham" is used today for a whole family of algorithms extending or modifying Bresenham's original algorithm. See further references below.

Contents

The algorithm

The common conventions that pixel coordinates increase in the down and right directions (e.g. that the pixel at (1,1) is directly above the pixel at (1,2)) and that the pixel centers that have integer coordinates will be used. The endpoints of the line are the pixels at (x_0, y_0) and (x_1, y_1), where the first coordinate of the pair is the column and the second is the row.

The algorithm will be initially presented only for the octant in which the segment goes down and to the right ($x_0 \le x_1$ and $y_0 \le y_1$), and its horizontal projection $x_1 - x_0$ is longer than the vertical projection $y_1 - y_0$ (the line has aslope whose absolute value is less than 1 and greater than 0.) In this octant, for each column x between x_0 and x_1, there is exactly one row y (computed by the algorithm) containing a pixel of the line, while each row between $y_0$and y_1 may contain multiple rasterized pixels.