LIST OF PRACTICALS

Note : All Practical's are performed using C language.

➢ **Practical 1(a) :** Study and enlist the basic functions used for graphics in C / C++ / Python language. Give an example for each of them.

Soln. :

1. arc

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, end angle 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.

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

void main()
{
  int gd = DETECT, gm;

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

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

  getch();
  closegraph();
}
```

2. bar

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.

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

void main()
{
  int gd = DETECT, gm;

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

  bar(100, 100, 200, 200);

  getch();
  closegraph();
}
```

3. bar3d

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.

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

void main()
{
  int gd = DETECT, gm;
```

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

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

  getch();
  closegraph();
}
```

4. 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.

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

void main()
{
  int gd = DETECT, gm;

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

  circle(100, 100, 50);

  getch();
  closegraph();
}
```

5. cleardevice

Declaration

```
void cleardevice();
```

cleardevice function clears the screen in graphics mode and sets the current position to (0, 0). Clearing the screen consists of filling the screen with current background color.

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

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

  outtext("Press any key to clear the screen.");
  getch();
  cleardevice();
  outtext("Press any key to exit...");

  getch();
  closegraph();
}
```

6. closegraph

closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

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

void main()
{
  int gd = DETECT, gm;

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

  outtext("Press any key to close the graphics mode...");

  getch();
  closegraph();
}
```

7. drawpoly

Declaration

```
void drawpoly( int num, int *polypoints );
```

Drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

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

void main()
{
  int gd=DETECT,gm,points[]={320,150,420,300,250,
300,320,150};

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

  drawpoly(4, points);
```

```
  getch();
  closegraph();
}
```

8. ellipse

Declaration

```
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.

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

void main()
{
  int gd = DETECT, gm;

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

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

  getch();
  closegraph();
}
```

9. fillellipse

Declaration

```
void fillellipse(int x, int y, int xradius, int yradius);
```

x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

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

void main()
{
  int gd = DETECT, gm;

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

  fillellipse(100, 100, 50, 25);

  getch();
  closegraph();
}
```

10. fillpoly

Declaration

```
void drawpoly( int num, int *polypoints );
```

fillpoly function draws and fills a polygon. It require same arguments as drawpoly.

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

void main()
{
  int gd=DETECT,gm,points[]={320,150,440,340,230,
340,320,150};

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

  fillpoly(4, points);
   getch();
  closegraph();
}
```

11. floodfill

Declaration

```
void floodfill(int x, int y, int border);
```

floodfill function is used to fill an enclosed area. Current fill pattern and fill color is used to fill the area (x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled, border specifies the color of boundary of area. To change fill pattern and fill color use setfillstyle. Code given below draws a circle and then fills it.

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

void main()
{
  int gd = DETECT, gm;

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

  setcolor(RED);
```

```
  circle(100,100,50);
  floodfill(100,100,RED);
   getch();
  closegraph();
}
```

12. getarccoords

Declaration

```
void getarccoords(struct arccoordstype *var);
```

getarccoords function is used to get coordinates of arc which is drawn most recently. arccoordstype is a predefined structure which is defined as follows :

```
struct arccoordstype
{
  int x, y;                 /*  center point of arc   */
  int xstart, ystart;    /*  start position        */
  int xend, yend;        /*  end position          */
};
```

address of a structure variable of type arccoordstype is passed to function getarccoords.

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

void main()
{
  int gd = DETECT, gm;
  struct arccoordstype a;
  char arr[100];

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

  arc(250,200,0,90,100);
  getarccoords(&a);

  sprintf(arr,"(%d, %d)",a.xstart,a.ystart);
  outtextxy(360,195,arr);

  sprintf(arr,"(%d, %d)",a.xend,a.yend);
  outtextxy(245,85,arr);

  getch();
  closegraph();
}
```

13. getbkcolor

Declaration

```
int getbkcolor();
```

getbkcolor function returns the current background color

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

void 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();
}
```

14. getcolor

Declaration

```
int getcolor();
```

getcolor function returns the current drawing color.

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

void 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();
}
```

15. getdrivername

getdrivername function returns a pointer to the current graphics driver.

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

void main()
{
  int gd = DETECT, gm;

  char *drivername;

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

  drivername = getdrivername();
  outtextxy(200, 200, drivername);

  getch();
  closegraph();

}
```

16. getimage

Declaration

```
void getimage(int left, int top, int right, int bottom, void *bitmap);
```

getimage function saves a bit image of specified region into memory, region can be any rectangle.

getimage copies an image from screen to memory. Left, top, right, and bottom define the area of the screen from which the rectangle is to be copied, bitmap points to the area in memory where the bit image is stored.

```
include<graphics.h>
#include<conio.h>
#include<stdlib.h>
 void main()
{
  int gd = DETECT, gm, area, temp1, temp2, left = 25, top = 75;
  void *p;

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

  setcolor(YELLOW);
  circle(50,100,25);
  setfillstyle(SOLID_FILL,YELLOW);
  floodfill(50,100,YELLOW);

  setcolor(BLACK);
  setfillstyle(SOLID_FILL,BLACK);
  fillellipse(44,85,2,6);
  fillellipse(56,85,2,6);

  ellipse(50,100,205,335,20,9);
  ellipse(50,100,205,335,20,10);
  ellipse(50,100,205,335,20,11);

  area = imagesize(left, top, left + 50, top + 50);
  p = malloc(area);

  setcolor(WHITE);
  settextstyle(SANS_SERIF_FONT,HORIZ_DIR,2);
  outtextxy(155,451,"Smiling Face Animation");

  setcolor(BLUE);
  rectangle(0,0,639,449);

  while(!kbhit())
  {
    temp1 = 1 + random ( 588 );
    temp2 = 1 + random ( 380 );

    getimage(left, top, left + 50, top + 50, p);
    putimage(left, top, p, XOR_PUT);
    putimage(temp1 , temp2, p, XOR_PUT);
    delay(100);
    left = temp1;
    top = temp2;
  }

  getch();
  closegraph();
}
```

17. getmaxcolor

Declaration

```
int getmaxcolor();
```

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.

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

void 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();

}
```

18. getmaxx

Declaration

```
int getmaxx();
```

getmaxx function returns the maximum X coordinate for current graphics mode and driver.

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

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

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

  max_x = getmaxx();

  sprintf(array, "Maximum X coordinate for current graphics mode and driver = %d.",max_x);
  outtext(array);

  getch();
  closegraph();
}
```

19. getmaxy

Declaration

```
int getmaxy();
```

getmaxy function returns the maximum Y coordinate for current graphics mode and driver.

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

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

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

  max_y = getmaxy();

  sprintf(array, "Maximum Y coordinate for current graphics mode and driver is = %d.",max_y);
  outtext(array);

  getch();
  closegraph();
}
```

20. getpixel

Declaration

```
int getpixel(int x, int y);
```

getpixel function returns the color of pixel present at location(x, y).

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

void main()
{
```

```
int gd = DETECT, gm, color;
char array[50];
initgraph(&gd,&gm,"C:\\TC\\BGI");

color = getpixel(0, 0);

sprintf(array,"color of pixel at (0,0) = %d",color);
outtext(array);

getch();
closegraph();
}
```

21. getx

Declaration

```
int getx();
```

getx function returns the X coordinate of current position.

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

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

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

  sprintf(array, "Current position of x = %d",getx());

  outtext(array);

  getch();
  closegraph();
}
```

22. gety

Declaration

```
int gety();
```

gety function returns the y coordinate of current position.

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

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

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

  y = gety();

  sprintf(array, "Current position of y = %d", y);

  outtext(array);

  getch();
  closegraph();

}
```

23. graphdefaults

Declaration

```
void graphdefaults();
```

graphdefaults function resets all graphics settings to their defaults. It resets the following graphics settings :

- Sets the viewport to the entire screen.
- Moves the current position to (0, 0).
- Sets the default palette colors, background color, and drawing color.
- Sets the default fill style and pattern.
- Sets the default text font and justification.

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

void main()
{
  int gd = DETECT, gm;

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

  setcolor(RED);
  setbkcolor(YELLOW);

  circle(250, 250, 50);

  getch();
```

```
  graphdefaults();

  getch();
  closegraph();

}
```

24. grapherrormsg

Declaration

```
char *grapherrormsg( int errorcode );
```

grapherrormsg function returns an error message string.

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

void main()
{
  int gd, gm, errorcode;

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

  errorcode = graphresult();

  if(errorcode != grOk)
  {
    printf("Graphics error: %s\n",
              grapherrormsg(errorcode));
    printf("Press any key to exit.");
    getch();
    exit(1);
  }

  getch();
  closegraph();
}
```

25. imagesize

Declaration

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

imagesize function returns the number of bytes required to store a bitimage. This function is used when we are using getimage.

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

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

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

  circle(200, 200, 50);
  line(150, 200, 250, 200);
  line(200, 150, 200, 250);

  bytes = imagesize(150, 150, 250, 250);
  sprintf(array, "Number of bytes required to store required area = %d", bytes);
  outtextxy(10, 280, array);

  getch();
  closegraph();
}
```

26. line

Declaration

```
void line(int x1, int y1, int x2, int y2);
```

line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line. The code given below draws a line.

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

void main()
{
  int gd = DETECT, gm;

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

  line(100, 100, 200, 200);

  getch();
  closegraph();
}
```

27. lineto

lineto function draws a line from current position(CP) to the point(x,y), you can get current position using getx and gety function.

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

void main()
{
  int gd = DETECT, gm;

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

  moveto(100, 100);
  lineto(200, 200);

  getch();
  closegraph();
}
```

28. linerel

linerel function draws a line from the current position(CP) to a point that is a relative distance (x, y) from the CP, then advances the CP by (x, y). You can use getx and gety to find the current position.

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

void main()
{
  int gd = DETECT, gm;

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

  moveto(250, 250);
  linerel(100, -100);

  getch();
  closegraph();
}
```

29. moveto

Declaration

```
void moveto(int x, int y);
```

moveto function changes the current position (CP) to (x, y)

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

void main()
{
  int gd = DETECT, gm;
  char msg[100];

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

  sprintf(msg, "X = %d, Y = %d",getx(),gety());

  outtext(msg);

  moveto(50, 50);

  sprintf(msg, "X = %d, Y = %d", getx(), gety());

  outtext(msg);

  getch();
  closegraph();
}
```

30. moverel

Declaration

```
void moverel(int x, int y);
```

moverel function moves the current position to a relative distance.

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

void 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();
}
```

31. outtext

Declaration

```
void outtext(char *string);
```

outtext function displays text at current position.

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

void main()
{
  int gd = DETECT, gm;

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

  outtext("To display text at a particular position on the screen use outtextxy");

  getch();
  closegraph();
}
```

32. pieslice

Declaration

```
void pieslice(int x, int y, int stangle, int endangle, int radius);
```

pieslice draws and fills a pie slice centered at (x,y) with a radius given by radius. The slice travels from stangle to endangle. The slice is outlined in the current drawing color and then filled using the current fill pattern and fill color.

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

void main()
{
  int gd = DETECT, gm;

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

  pieslice(200, 200, 0, 135, 100);

  getch();
  closegraph();
}
```

33. putimage

Declaration

```
void putimage(int left, int top, void *ptr, int op);
```

putimage function outputs a bit image onto the screen.

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

void main()
{
  int gd = DETECT, gm, area, temp1, temp2, left = 25, top = 75;
  void *p;

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

  setcolor(YELLOW);
  circle(50,100,25);
  setfillstyle(SOLID_FILL,YELLOW);
  floodfill(50,100,YELLOW);

  setcolor(BLACK);
  setfillstyle(SOLID_FILL,BLACK);
  fillellipse(44,85,2,6);
  fillellipse(56,85,2,6);

  ellipse(50,100,205,335,20,9);
  ellipse(50,100,205,335,20,10);
  ellipse(50,100,205,335,20,11);

  area = imagesize(left, top, left + 50, top + 50);
  p = malloc(area);
```

```
  setcolor(WHITE);
  settextstyle(SANS_SERIF_FONT,HORIZ_DIR,2);
  outtextxy(155,451,"Smiling Face Animation");

  setcolor(BLUE);
  rectangle(0,0,639,449);

  while(!kbhit())
  {
    temp1 = 1 + random ( 588 );
    temp2 = 1 + random ( 380 );

    getimage(left, top, left + 50, top + 50, p);
    putimage(left, top, p, XOR_PUT);
    putimage(temp1 , temp2, p, XOR_PUT);
    delay(100);
    left = temp1;
    top = temp2;
  }

  getch();
  closegraph();
}
```

34. putpixel

Declaration

```
void putpixel(int x, int y, int color);
```

putpixel function plots a pixel at location (x, y) of specified color.

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

void main()
{
  int gd = DETECT, gm;

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

  putpixel(25, 25, RED);

  getch();
  closegraph();
}
```

35. rectangle

Declaration

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

rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. 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. The code given below draws a rectangle.

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

void main()
{
  int gd = DETECT, gm;

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

  rectangle(100,100,200,200);

  getch();
  closegraph();
}
```

36. sector

Declaration

```
void sector( int x, int y, int stangle, int endangle, int xradius, int yradius);
```

Sector function draws and fills an elliptical pie slice.

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

void main()

{
  int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
   sector(100, 100, 0, 135, 25, 35);
   getch();
  closegraph();
}
```

37. setbkcolor

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.

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

void 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();
}
```

38. setcolor

Declaration

void setcolor(int color);

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.

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

void 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();
}
```

39. setfillstyle

Declaration

void setfillstyle(int pattern, int color);

setfillstyle function sets the current fill pattern and fill 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
};

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

void 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();
}
```

40. setlinestyle

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
};
```

setlinestyle sets the style for all lines drawn by line, lineto, rectangle, drawpoly, and so on.

The linesettingstype structure is defined in graphics.h as follows :

```
struct linesettingstype
{
 int linestyle;
 unsigned upattern;
 int thickness;
};
```

```
#include <graphics.h>

void 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();
}
```

41. settextstyle

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).

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.

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
};
```

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

void 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();
}
```

42. setviewport

Declaration

```
void setviewport(int left, int top, int right, int bottom, int clip);
```

setviewport function sets the current viewport for graphics output.

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

void 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();
}
```

43. textheight

Declaration

```
int textheight(char *string);
```

textheight function returns the height of a string in pixels.

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

void 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();
}
```

44. textwidth

Declaration

```
int textwidth(char *string);
```

textwidth function returns the width of a string in pixels.

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

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

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

  width = textwidth("C programming");

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

  getch();
  closegraph();
}
```

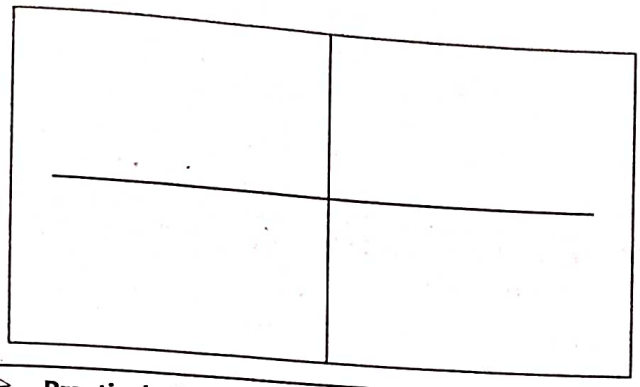
➢ **Practical 1(b) :** Draw a co-ordinate axis at the center of the screen.

Soln. :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
    int gd=DETECT,gm;
    int midx,midy;
    initgraph(&gd,&gm,"\\TURBOC3\\bgi");
    cleardevice();
    midx=getmaxx()/2;
    midy=getmaxy()/2;
    //coordinate Axes for default 480x640 window
    line(1,midy,640,midy);
    line(midx,1,midx,480);
```

```
    getch();

}
```

Output

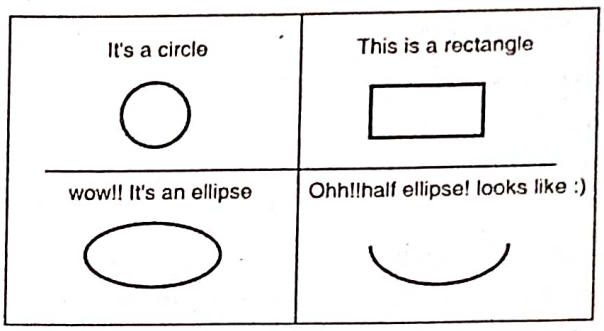


➢ **Practical 2(a) :** Divide your screen into four region, draw circle, rectangle, ellipse and half ellipse in each region with appropriate message.

Soln. :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm;
int midx,midy;
initgraph(&gd,&gm,"C:\\TURBOC3\\bgi");
cleardevice();
midx=getmaxx()/2;
midy=getmaxy()/2;
//coordinate Axis
line(1,midy,640,midy);
line(midx,1,midx,480);
setcolor(RED);
circle(midx+(-150),midy-(120),40);
printf("\t\tIt's a circle ");
setcolor(GREEN);
rectangle(midx+(100),midy-(100),midx+(200),midy-
(150));
printf("\t\t\t\t This is Rectangle\n\n\n\n\n");
setcolor(BLUE);
ellipse(midx+(-150),midy-(-100),0,360,midx+
(-250),midy-(200));
printf("\n\n\n\n\n\n\n\n\n\n\n\t wow!! It's an
elllpse");
setcolor(YELLOW);
ellipse(midx+(180),midy-(-100),180,0,midx+
(-200),midy-(150));
printf("\t\t\t Ohh!!half ellipse! looks like :)");
getch();
}
```

Output



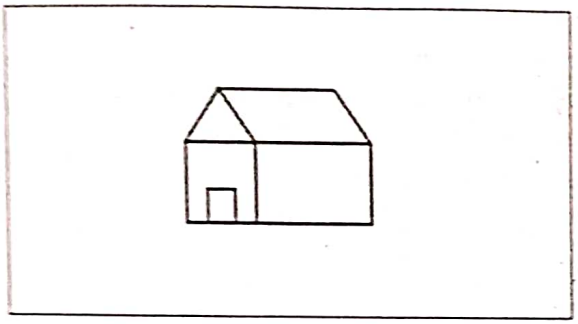
➢ **Practical 2(b) :** Draw a simple hut on the screen.

Soln. :

1. Simple hut

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

void main()
{
   int gd = DETECT,gm;
  initgraph(&gd, &gm, "c:\\turboc3\\BGI");
  /* Draw Hut */
  setcolor(WHITE);
  rectangle(150,180,250,300);
  rectangle(250,180,420,300);
  rectangle(180,250,220,300);

  line(200,100,150,180);
  line(200,100,250,180);
  line(200,100,370,100);
  line(370,100,420,180);
  getch();
  closegraph();
}
```

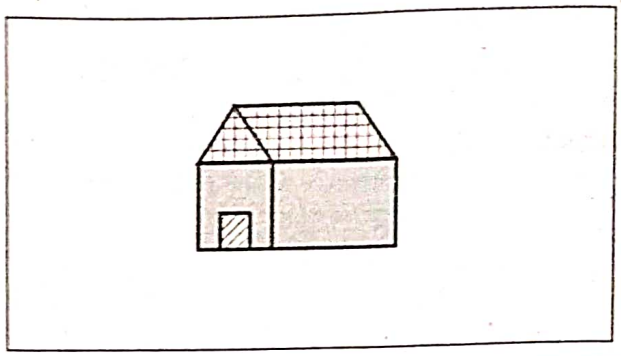
Output



2. Colorful hut

```
#include<graphics.h>
#include<conio.h>
void main()
{
 int gd = DETECT,gm;
   initgraph(&gd, &gm, "C:\\turboc3\\BGI");
   /* Draw Hut */
   setcolor(WHITE);
   rectangle(150,180,250,300);
   rectangle(250,180,420,300);
   rectangle(180,250,220,300);

   line(200,100,150,180);
   line(200,100,250,180);
   line(200,100,370,100);
   line(370,100,420,180);

   /* Fill colours */
   setfillstyle(SOLID_FILL, BROWN);
   floodfill(152, 182, WHITE);
   floodfill(252, 182, WHITE);
   setfillstyle(SLASH_FILL, BLUE);
   floodfill(182, 252, WHITE);
   setfillstyle(HATCH_FILL, GREEN);
   floodfill(200, 105, WHITE);
   floodfill(210, 105, WHITE);

   getch();
   closegraph();
}
```

Output



➢ **Practical 3 :** Draw the following basic shapes in the center of the screen

(i) Circle (ii) Rectangle

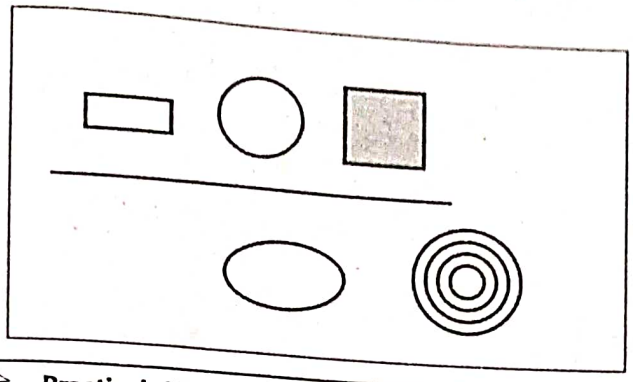
(iii) Square (iv) Concentric Circles

(v) Ellipse (vi) Line.

Soln. :

```
#include<graphics.h>
#include<conio.h>
void main()
{
  int gd = DETECT,gm,left=100,top=100,right=200,
bottom=200,x= 300,y=150,radius=50;
  initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

  rectangle(120, 150, 230, 200);
  circle(x, y, radius);

  //to draw square
  bar(left + 300, top, right + 300, bottom);
  line(left - 10, top + 150, left + 410, top + 150);

  ellipse(x, y + 200, 0, 360, 80, 50);
  //to draw concentric circle

 for (radius = 25; radius <= 100 ; radius = radius + 20)
   circle(500,350,radius);

  getch();
  closegraph();
 }
```

Output



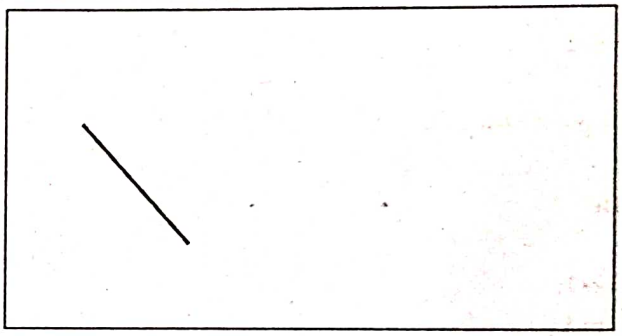
➢ **Practical 4(a) :** Develop the program for DDA Line drawing algorithm.

Soln. :

```
#include <graphics.h>
#include <stdio.h>
#include <math.h>

void main( )
{
  float x,y,x1,y1,x2,y2,dx,dy,pixel;
  int i,gd,gm;

  printf("Enter the value of x1 : ");
  scanf("%f",&x1);
  printf("Enter the value of y1 : ");
  scanf("%f",&y1);
  printf("Enter the value of x2 : ");
  scanf("%f",&x2);
  printf("Enter the value of y2 : ");
  scanf("%f",&y2);

  detectgraph(&gd,&gm);
  initgraph(&gd,&gm,"c:\\turboc3\\bgi"); // in this , copy paste your turboC bgi path

  dx=abs(x2-x1);
  dy=abs(y2-y1);

  if(dx>=dy)
  pixel=dx;
  else
  pixel=dy;

  dx=dx/pixel;
  dy=dy/pixel;

  x=x1;
  y=y1;
  i=1;
  while(i<=pixel)
  {
    putpixel(x,y,1);
    x=x+dx;
    y=y+dy;
    i=i+1;
    delay(100);
  }
  getch();
  closegraph();
}
```

Output

```
Enter the value of x1 : 100
Enter the value of y1 : 200
Enter the value of x2 : 300
Enter the value of y2 : 400
```



➢ **Practical 4(b) :** Develop the program for Bresenham's Line drawing algorithm.

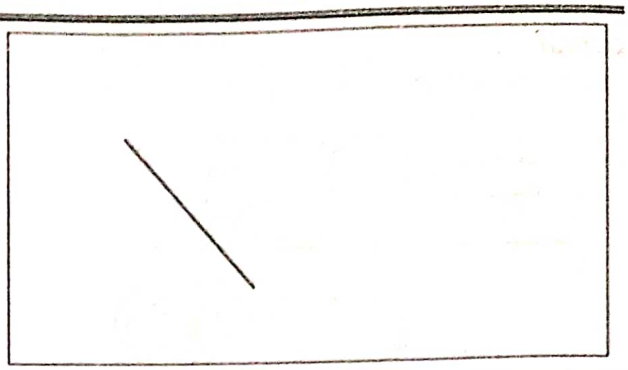
Soln. :

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

void main()
{
int dx,dy,x,y,p,x1,y1,x2,y2;
int gd,gm;
clrscr();
printf("\n\n\tEnter the co-ordinates of first point : ");
```

```
scanf("%d %d",&x1,&y1);
printf("\n\n\tEnter the co-ordinates of second point : ");
scanf("%d %d",&x2,&y2);

dx = (x2 - x1);
dy = (y2 - y1);
p = 2 * (dy) - (dx);

x = x1;
y = y1;

detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\turboc3\\bgi"); // in this , copy paste your turboC bgi path
putpixel(x,y,WHITE);

while(x <= x2)
{
if(p < 0)
{
x=x+1;
y=y;
p = p + 2 * (dy);
}
else
{
x=x+1;
y=y+1;
p = p + 2 * (dy - dx);
}
putpixel(x,y,WHITE);
}
getch();
closegraph();
}
```

Output

```
Enter the co-ordinates of first point : 100
200
Enter the co-ordinates of second point : 300
400
```



➢ **Practical 5(a) :** Develop the program for the mid-point circle drawing algorithm.

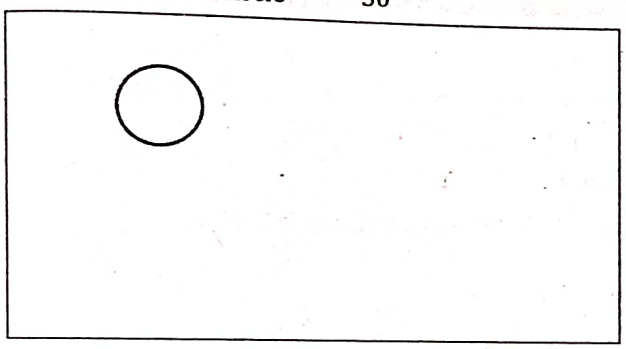
Soln. :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void pixel(int xc,int yc,int x,int y);
void main()
{
    int gd=DETECT,gm,xc,yc,r,x,y,Pk;
    clrscr();
    initgraph(&gd,&gm,"c:\\turboc3\\bgi ");
    printf("*** Bresenham's Midpoint algorithm of circle ***\n");
    printf("Enter the value of Xc\t");
    scanf("%d",&xc);
    printf("Enter the value of Yc \t");
    scanf("%d",&yc);
    printf("Enter the Radius of circle\t");
    scanf("%d",&r);
    x=0;
    y=r;
    Pk=1-r;
    pixel(xc,yc,x,y);
    while(x<y)
    {
        if(Pk<0)
        {
            x=x+1;
            Pk=Pk+(2*x)+1;
        }
        else
        {
            x=x+1;
            y=y-1;
            Pk=Pk+(2*x)-(2*y)+1;
```

```
            }
            pixel(xc,yc,x,y);
        }
        getch();
        closegraph();
}
void pixel(int xc,int yc,int x,int y)
{
        putpixel(xc+x,yc+y,7);
        putpixel(xc+y,yc+x,7);
        putpixel(xc-y,yc+x,7);
        putpixel(xc-x,yc+y,7);
        putpixel(xc-x,yc-y,7);
        putpixel(xc-y,yc-x,7);
        putpixel(xc+y,yc-x,7);
        putpixel(xc+x,yc-y,7);
}
```

Output

```
*** Bresenham's Midpoint algorithm of circle ***
Enter the value of Xc          100
Enter the value of Yc          200
Enter the Radius of Circle     50
```



➢ **Practical 5(b) :** Develop the program for the mid-point ellipse drawing algorithm.

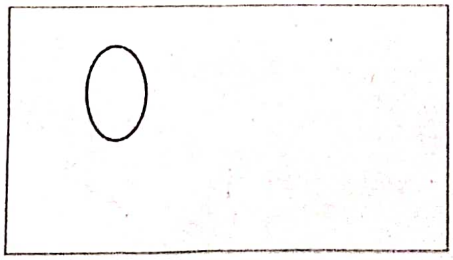
Soln. :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void disp();
float x,y;
int xc,yc;
void main()
{
int gd=DETECT,gm;
int a,b;
float p1,p2;
clrscr();
initgraph(&gd,&gm,"c://turboc3//bgi");
printf("Enter xc:\t");
scanf("%d",&xc);
printf("Enter yc:\t");
scanf("%d",&yc);
printf("Enter a:\t");
scanf("%d",&a);
printf("Enter b:\t");
scanf("%d",&b);
x=0;y=b;
disp();
p1=(b*b)-(a*a*b)+(a*a)/4;
while((2.0*b*b*x)<=(2.0*a*a*y))
{
x++;
if(p1<=0)
p1=p1+(2.0*b*b*x)+(b*b);
else
{
y--;
p1=p1+(2.0*b*b*x)+(b*b)-(2.0*a*a*y);
}
disp();
x=-x;
disp();
x=-x;
}
x=a;
y=0;
disp();
```

```
p2=(a*a)+2.0*(b*b*a)+(b*b)/4;
while((2.0*b*b*x)>(2.0*a*a*y))
-{
y++;
if(p2>0)
p2=p2+(a*a)-(2.0*a*a*y);
else
{
x--;
p2=p2+(2.0*b*b*x)-(2.0*a*a*y)+(a*a);
}
disp();
y=-y;
disp();
y=-y;
}
getch();
closegraph();
}
void disp()
{
putpixel(xc+x,yc+y,10);
putpixel(xc-x,yc+y,10);
putpixel(xc+x,yc-y,10);
putpixel(xc+x,yc-y,10);
}
```

Output

```
Enter xc :100
Enter yc : 200
Enter a : 30
Enter b : 60
```



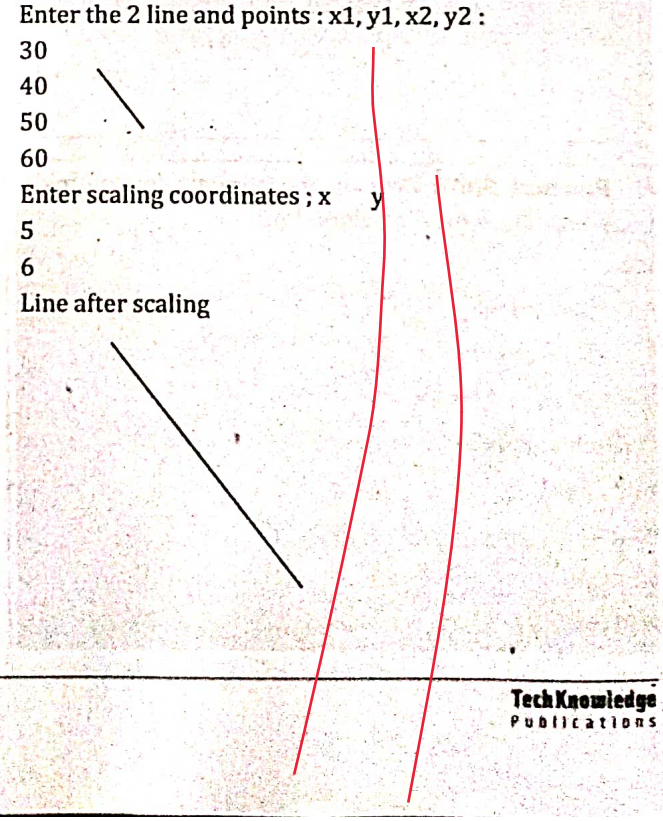
➢ **Practical 6(a) :** Write a program to implement 2D scaling.

Soln. :

```
#include<graphics.h>
#include<stdio.h>
void main()
{
int i;
int gd=DETECT,gm;
int x2,y2,x1,y1,x,y;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
printf("Enter the 2 line end points: x1,y1,x2,y2:\n");
scanf("%d\n%d\n%d\n%d",&x1,&y1,&x2,&y2);
line(x1,y1,x2,y2);

printf("\nEnter scaling co-ordinates;x\t y\t \n ");
scanf("%d%d",&x,&y);
x1=(x1*x);
y1=(y1*y);
x2=(x2*x);
y2=(y2*y);
printf("Line after scaling");
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

Output

```
Enter the 2 line and points : x1, y1, x2, y2 :
30
40
50
60
Enter scaling coordinates ; x     y
5
6
Line after scaling
```



➢ **Practical 6(b) :** Write a program to perform 2D translation

Soln. :

```
#include<graphics.h>
#include<stdio.h>
void main()
{
int i;
int gd=DETECT,gm;
int x2,y2,x1,y1,x,y;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
printf("Enter the 2 line end points: x1,y1,x2,y2:\n");
scanf("%d\n%d\n%d\n%d",&x1,&y1,&x2,&y2);
line(x1,y1,x2,y2);

printf("\nEnter scaling co-ordinates;x\t y\t \n ");
scanf("%d%d",&x,&y);

x1=x1+x;
y1=y1+y;
x2=x2+x;
y2=y2+y;
printf("Line after translation");
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

Output

```
Enter the 2 line end points : x1, y1, x2, y2 :
30
40
50
60
Enter scaling coordinates ; x      y
100
200
Line after translation
```

➢ **Practical 7(a) :** Perform 2D Rotation on a given object.

Soln. :

```
#include<graphics.h>
#include<stdio.h>
void main()
{
int gd=DETECT,gm;
int i;
int x2,y2,x1,y1,x,y,xn,yn;
double r11,r12,r21,r22,th;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
printf("Enter the 2 line end points x1,y1,x2,y2: \n");
scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

line(x1,y1,x2,y2);

printf("\n\n\n Enter the angle:\t");
scanf("%lf",&th);
r11=cos((th*3.1428)/180);
r12=sin((th*3.1428)/180);
r21=(-sin((th*3.1428)/180));
r22=cos((th*3.1428)/180);
xn=((x2*r11)-(y2*r21));
yn=((x2*r12)+(y2*r22));
line(x1,y1,xn,yn);
getch();
closegraph();
}
```

Output

```
Enter the 2 line end points x1, y1, x2, y2 :
100
150
40
30
```

Enter the angle

Enter the 2 line end points x1, y1, x2, y2 :
100
150
40
30

Enter the angle : 45

➢ **Practical 7(b) :** Program to create a house like figure and perform the following operations.
i. Scaling about the origin followed by translation.
ii. Scaling with reference to an arbitrary point.
iii. Reflect about the line y = mx + c.

Soln. :

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

void reset (int h[][2])
{
  int val[9][2] = {
                { 50, 50 },{ 75, 50 },{ 75, 75 },{ 100, 75 },
                { 100, 50 },{ 125, 50 },{ 125, 100 },
                { 87, 125 },{ 50, 100 }
              };
  int i;
  for (i=0; i<9; i++)
  {
    h[i][0] = val[i][0]-50;
    h[i][1] = val[i][1]-50;
  }
}
void draw (int h[][2])
{
  int i;
  setlinestyle (DOTTED_LINE, 0, 1);
  line (320, 0, 320, 480);
  line (0, 240, 640, 240);
  setlinestyle (SOLID_LINE, 0, 1);
  for (i=0; i<8; i++)
    line (320+h[i][0], 240-h[i][1], 320+h[i+1][0], 240-
h[i+1][1]);
  line (320+h[0][0], 240-h[0][1], 320+h[8][0], 240-
h[8][1]);
}
void rotate (int h[][2], float angle)
{
  int i;
  for (i=0; i<9; i++)
  {
    int xnew, ynew;
    xnew = h[i][0] * cos (angle) - h[i][1] * sin (angle);
    ynew = h[i][0] * sin (angle) + h[i][1] * cos (angle);
    h[i][0] = xnew; h[i][1] = ynew;
  }
}
void scale (int h[][2], int sx, int sy)
{
  int i;
  for (i=0; i<9; i++)
  {
    h[i][0] *= sx;
    h[i][1] *= sy;
  }
}
void translate (int h[][2], int dx, int dy)
{
  int i;
  for (i=0; i<9; i++)
  {
    h[i][0] += dx;
    h[i][1] += dy;
  }
}
void reflect (int h[][2], int m, int c)
{
    int i;
```

```
    float angle;
    for (i=0; i<9; i++)
        h[i][1] -= c;
    angle = M_PI/2 - atan (m);
    rotate (h, angle);
    for (i=0; i<9; i++)
        h[i][0] = -h[i][0];
    angle = -angle;
    rotate (h, angle);
    for (i=0; i<9; i++)
        h[i][1] += c;
}

void ini()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"..\\bgi");
}
void dini()
{
    getch();
    closegraph();
}
void main()
{

    int h[9][2],sx,sy,x,y,m,c,choice;
    do
    {
        clrscr();
        printf("1. Scaling about the origin.\n");
        printf("2. Scaling about an arbitrary point.\n");
        printf("3. Reflection about the line y = mx +
                c.\n");
        printf("4. Exit\n");
        printf("Enter the choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf ("Enter the x- and y-scaling
                factors: ");
                scanf ("%d%d", &sx, &sy);
                ini();
                reset (h);
                draw (h);getch();
                scale (h, sx, sy);
                cleardevice();
                draw (h);
                dini();
                break;

            case 2: printf ("Enter the x- and y-scaling
                factors: ");
                scanf ("%d%d", &sx, &sy);
                printf ("Enter the x- and y-coordinates
                of the point: ");
                scanf ("%d%d", &x, &y);
                ini();
                reset (h);
                translate (h, x, y);
                        // Go to arbitrary point
                draw(h); getch();
                        //Show its arbitrary position
                cleardevice();
                translate(h,-x,-y);
                        //Take it back to origin
                draw(h);
                getch();
                cleardevice();
                scale (h, sx, sy);//Now Scale it
                draw(h);
                getch();
                translate (h, x, y);
                        //Back to Arbitrary point
                cleardevice();
                draw (h);
                putpixel (320+x, 240-y, WHITE);
                dini();
                break;

            case 3: printf ("Enter the values of m and c: ");
                scanf ("%d%d", &m, &c);
                ini();
                reset (h);
                draw (h); getch();
                reflect (h, m, c);
                cleardevice();
                draw (h);
                dini();
                break;
```

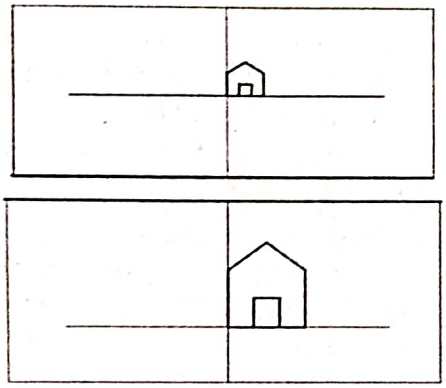
```
            case 4: exit(0);
        }
    }while(choice!=4);
}
```

Output

1. Scaling about the origin.
2. Scaling about an arbitrary point.
3. Reflection about the line y = mx + c
4. Exit

Enter the choice : 1

Enter the x- and y-scaling factors : 2
3

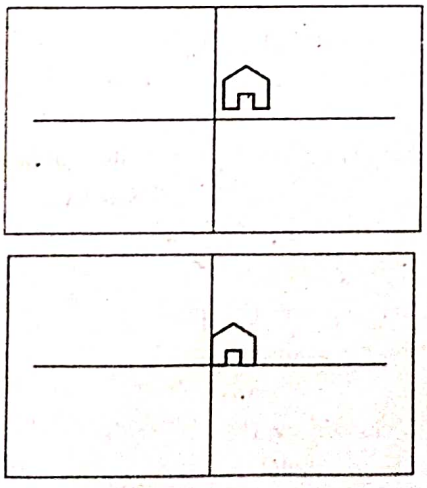


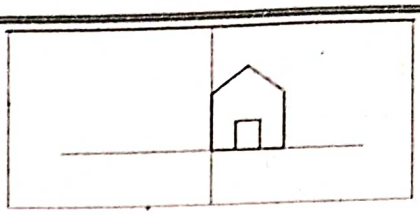
1. Scaling about the origin.
2. Scaling about an arbitrary point.
3. Reflection about the line y = mx + c
4. Exit

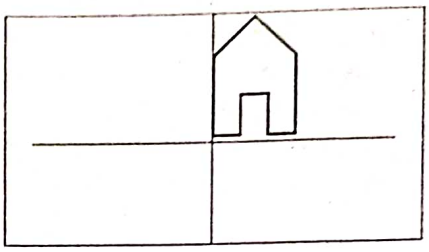
Enter the choice : 2

Enter the x- and y-scaling factors : 2 3

Enter the x- and y-coordinates of the point : 3 10_



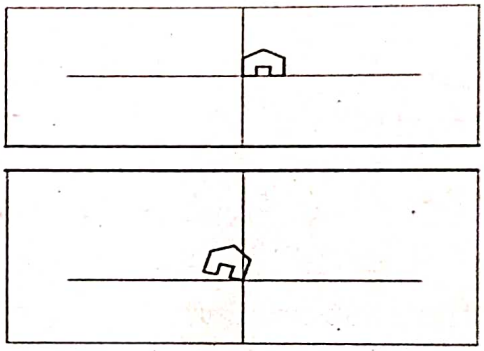




1. Scaling about the origin.
2. Scaling about an arbitrary point.
3. Reflection about the line y = mx + c
4. Exit

Enter the choice : 3

Enter the values of m and : 5 10



➢ **Practical 8(a) :** Write a program to implement Cohen-Sutherland clipping.

Soln. :

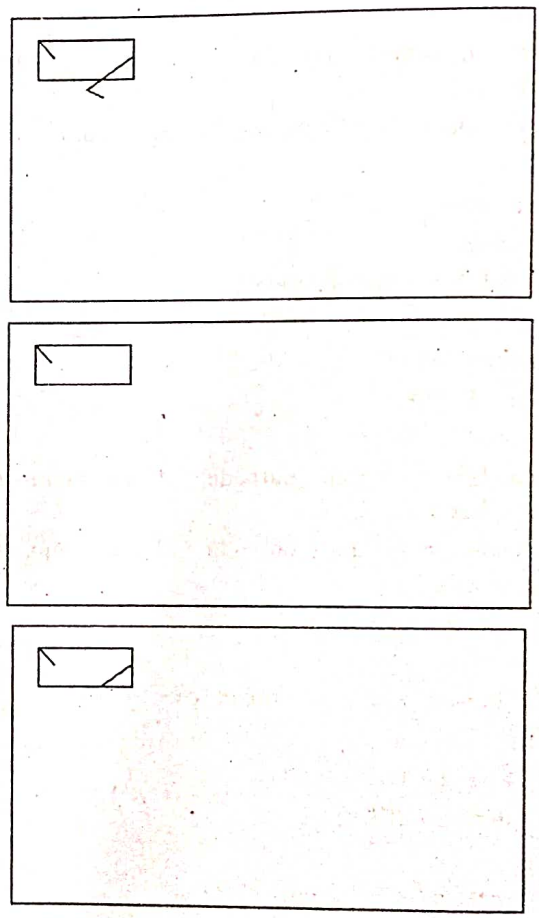
```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#define MAX 20

enum { TOP = 0x1, BOTTOM = 0x2, RIGHT = 0x4, LEFT =
0x8 };

enum { FALSE, TRUE };
typedef unsigned int outcode;

outcode compute_outcode(int x, int y,
            int xmin, int ymin, int xmax, int ymax)
{
```

```
  outcode oc = 0;

  if (y > ymax)
    oc = TOP;
  else if (y < ymin)
    oc = BOTTOM;

  if (x > xmax)
    oc = RIGHT;
  else if (x < xmin)
    oc = LEFT;

  return oc;
}

void cohen_sutherland (double x1, double y1, double x2,
double y2,...
double xmin, double ymin, double xmax, double ymax)
{
  int accept;
  int done;
  outcode outcode1, outcode2;

  accept = FALSE;
  done = FALSE;

  outcode1 = compute_outcode (x1, y1, xmin, ymin,
xmax, ymax);
  outcode2 = compute_outcode (x2, y2, xmin, ymin,
xmax, ymax);
  do
  {
    if (outcode1 == 0 && outcode2 == 0)
    {
      accept = TRUE;
      done = TRUE;
    }
    else if (outcode1 & outcode2)
    {
      done = TRUE;
    }
    else
    {
      double x, y;
      int outcode_ex = outcode1 ? outcode1 : outcode2;
      if (outcode_ex & TOP)
      {
        x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
        y = ymax;
      }

      else if (outcode_ex & BOTTOM)
      {
        x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
        y = ymin;
      }
      else if (outcode_ex & RIGHT)
      {
        y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
        x = xmax;
      }
      else
      {
        y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
        x = xmin;
      }
      if (outcode_ex == outcode1)
      {
        x1 = x;
        y1 = y;
        outcode1 = compute_outcode (x1, y1, xmin,
ymin, xmax, ymax);
      }
      else
      {
        x2 = x;
        y2 = y;
        outcode2 = compute_outcode (x2, y2, xmin,
ymin, xmax, ymax);
      }
    }
  } while (done == FALSE);

  if (accept == TRUE)
    line (x1, y1, x2, y2);
}
void main()
{
  int n;
  int i, j;
```

```
  int ln[MAX][4];
  int clip[4];
  int gd = DETECT, gm;
  clrscr();

  printf ("Enter the number of lines to be clipped:\n");
  scanf ("%d", &n);

  printf ("Enter the x- and y-coordinates of the line-endpoints:\n");
  for (i=0; i<n; i++)
    for (j=0; j<4; j++)
      scanf ("%d", &ln[i][j]);

  printf ("Enter the x- and y-coordinates of the left-top and right-");
  printf ("bottom corners\nof the clip window:\n");
  for (i=0; i<4; i++)
    scanf ("%d", &clip[i]);

  initgraph(&gd, &gm, "C:\\turboc3\\bgi");

  rectangle(clip[0], clip[1], clip[2], clip[3]);
  for (i=0; i<n; i++)
    line (ln[i][0], ln[i][1], ln[i][2], ln[i][3]);
  getch();
  cleardevice();
  rectangle (clip[0], clip[1], clip[2], clip[3]);
  for (i=0; i<n; i++)
  {
    cohen_sutherland (ln[i][0], ln[i][1], ln[i][2], ln[i][3],
      clip[0], clip[1], clip[2], clip[3]);
    getch();
  }
  closegraph();
}
```

Output

```
Enter the number of lines to be clipped :
3
Enter the x- and y-coordinates of the line –endpoints :
10
20
30
40
50
60
70
80
90
30
50
60
Enter the x- and y-coordinates of the left-top and right-bottom corners of the clip window :
10
20
90
50
```



➢ **Practical 8(b) :** Write a program to implement Liang - Barsky Line Clipping Algorithm.

Soln. :

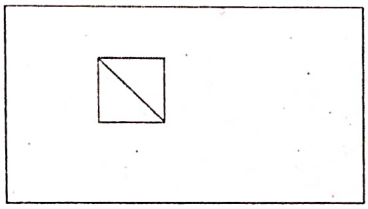
```
#include<stdio.h>
#include<graphics.h>
```

```
void main()
{
  int i,gd=DETECT,gm;
  int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,
dx,dy;
  float t1,t2,p[4],q[4],temp;
  clrscr();
  printf("Enter line coordinates x1 , y1:");
  scanf("%d %d",&x1,&y1);

  printf("Enter line coordinates x2 , y2:");
  scanf("%d %d",&x2,&y2);

  xmin=100;
  ymin=100;
  xmax=250;
  ymax=250;

  initgraph(&gd,&gm,"c:\\turboc3\\bgi");
  rectangle(xmin, ymin, xmax, ymax);
  dx=x2-x1;
  dy=y2-y1;

  p[0]=-dx;
  p[1]=dx;
  p[2]=-dy;
  p[3]=dy;

  q[0]=x1-xmin;
  q[1]=xmax-x1;
  q[2]=y1-ymin;
  q[3]=ymax-y1;

  for(i=0;i<4;i++)
  {
    if(p[i]==0)
    {
      printf("line is parallel to one of the clipping
boundary");
      if(q[i]>=0)
      {
        if(i<2)
        {
          if(y1<ymin)
          {
            y1=ymin;
          }

          if(y2>ymax)
          {
            y2=ymax;
          }

          line(x1,y1,x2,y2);
        }

        if(i>1)
        {
          if(x1<xmin)
          {
            x1=xmin;
          }

          if(x2>xmax)
          {
            x2=xmax;
          }

          line(x1,y1,x2,y2);
        }
      }
    }
  }

  t1=0;
  t2=1;

  for(i=0;i<4;i++)
  {
    temp=q[i]/p[i];

    if(p[i]<0)
    {
      if(t1<=temp)
        t1=temp;
    }
    else
    {
      if(t2>temp)
        t2=temp;
```

```
    }
  }

  if(t1<t2)
  {
    xx1 = x1 + t1 * p[1];
    xx2 = x1 + t2 * p[1];
    yy1 = y1 + t1 * p[3];
    yy2 = y1 + t2 * p[3];
    line(xx1,yy1,xx2,yy2);
  }
  delay(5000);
  closegraph();
}
```

Output

```
Enter line coordinates x1 ; y1 : 100
100
Enter line coordinates x2 , y2 : 300
300_
```



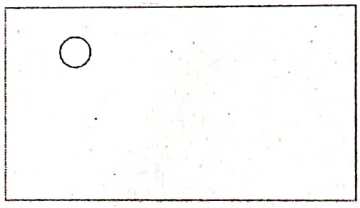
➢ **Practical 9(a) :** Write a program to fill a circle using Flood Fill Algorithm.

Soln. :

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

void floodFill(int x,int y,int oldcolor,int newcolor)
{
  if(getpixel(x,y) == oldcolor)
  {
    putpixel(x,y,newcolor);
    floodFill(x+1,y,oldcolor,newcolor);
    floodFill(x,y+1,oldcolor,newcolor);
    floodFill(x-1,y,oldcolor,newcolor);
    floodFill(x,y-1,oldcolor,newcolor);
  }
}
//getpixel(x,y) gives the color of specified pixel
void main()
{
  int gm,gd=DETECT,radius;
  int x,y;

  printf("Enter x and y positions for circle\n");
  scanf("%d%d",&x,&y);
  printf("Enter radius of circle\n");
  scanf("%d",&radius);

  initgraph(&gd,&gm,"c:\\turboc3\\bgi");
  circle(x,y,radius);
  floodFill(x,y,0,15);
  delay(5000);
  closegraph();

}
```

Output

```
Enter x and y positions for circle
100
100
Enter radius of circle
25
```



➢ **Practical 9(b) :** Write a program to fill a circle using Boundary Fill Algorithm.

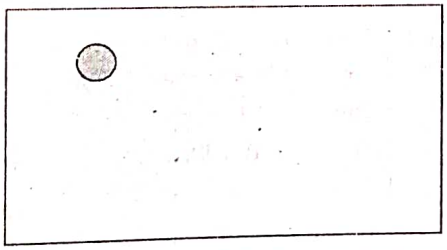
Soln. :

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

void boundaryfill(int x,int y,int f_color,int b_color)
{
  if(getpixel(x,y)!=b_color && getpixel(x,y)!=f_color)
  {
    putpixel(x,y,f_color);
    boundaryfill(x+1,y,f_color,b_color);
    boundaryfill(x,y+1,f_color,b_color);
    boundaryfill(x-1,y,f_color,b_color);
    boundaryfill(x,y-1,f_color,b_color);
```

```
  }
}
//getpixel(x,y) gives the color of specified pixel

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

  printf("Enter x and y positions for circle\n");
  scanf("%d%d",&x,&y);
  printf("Enter radius of circle\n");
  scanf("%d",&radius);

  initgraph(&gd,&gm,"c:\\turboc3\\bgi");
  circle(x,y,radius);
  boundaryfill(x,y,4,15);
  delay(5000);
  closegraph();

  }
```

Output

```
Enter x and y position for circle
100
100
Enter radius of circle
25
```



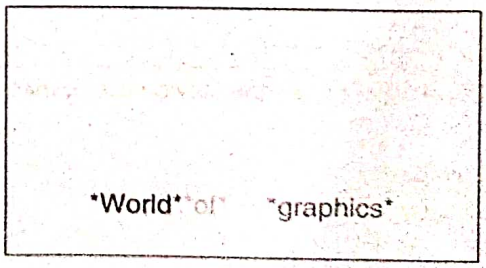
Note : The output will display circle filled with red color.

➢ **Practical 10(a) :** Develop a simple text screen saver using graphics functions.

Soln. :

```
//program for font animation//

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

void main()
{
int gd=DETECT,gm,x=600,i;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");

//setting limit of movement of text screen saver here we are setting it as 200 means it will move upto 250 times , if we increase this it will run in output screen upto that limit, thus we won't be able to come out of output screen which indicates it's still under execution not in infinite loop kindly note.
for(x=0;x<250;x++)
{
 x %= 250;
 setcolor(random(16));

 circle(random(635),random(70),50);
 circle(random(635),random(70),50);
 circle(random(635),random(70),50);
 circle(random(635),random(70),50);
 circle(random(635),random(70),50);

 clearviewport();

 settextstyle(1,0,5);
 setcolor(RED);
 outtextxy(50,415-2*x,"*World*");
 setcolor(GREEN);
 outtextxy(200,415-2*x,"*of*");
 setcolor(YELLOW);
 settextstyle(3,0,5);
 outtextxy(350,415-2*x,"*graphics*");
}
getch();
}
```

Output

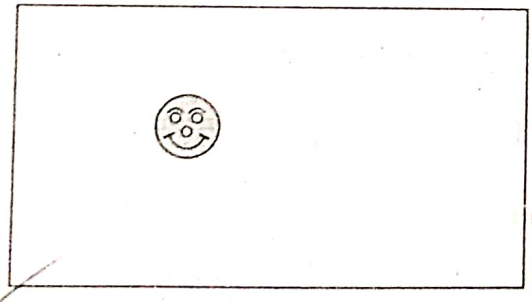


➢ **Practical 10(b) :** Perform smiling face animation using graphic functions.

Soln. :

```
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
void main()
{
 int gd = DETECT,gm;
 initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
//for head
 circle(200,200,30);
//for left eye
 circle(190,190,5);
 arc(190,190,50,130,10);
//for right eye
 circle(210,190,5);
 arc(210,190,50,130,10);
//for smiley lips
 arc(200,210,180,360,10);
 line(187,210,193,210);
 line(207,210,213,210);
//for nose
 line(198,195,195,200);
 line(202,195,205,200);
 line(195,200,200,205);
 line(205,200,200,205);
 getch();
 closegraph();
}
```

Output



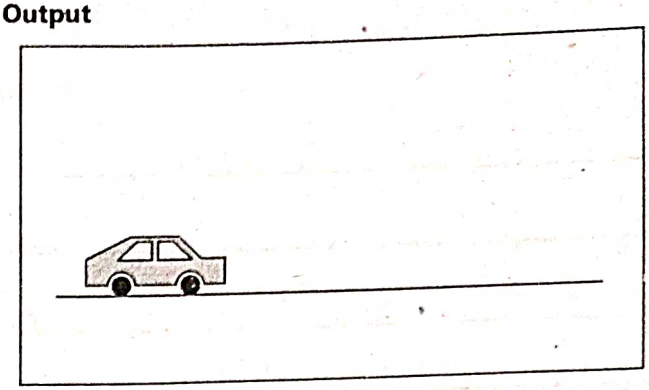
➢ **Practical 10(c) :** Draw the moving car on the screen.

Soln. :

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

  /* initialize graphic mode */
  initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
  /* maximum pixel in horizontal axis */
  maxx = getmaxx();
  /* mid pixel in vertical axis */
  midy = getmaxy()/2;

  for (i=0; i < maxx-150; i=i+5) {
    /* clears screen */
    cleardevice();

    /* draw a white road */
    setcolor(WHITE);
    line(0, midy + 37, maxx, midy + 37);

    /* Draw Car */
    setcolor(YELLOW);
    setfillstyle(SOLID_FILL, RED);

    line(i, midy + 23, i, midy);
    line(i, midy, 40 + i, midy - 20);
    line(40 + i, midy - 20, 80 + i, midy - 20);
    line(80 + i, midy - 20, 100 + i, midy);
    line(100 + i, midy, 120 + i, midy);
    line(120 + i, midy, 120 + i, midy + 23);
    line(0 + i, midy + 23, 18 + i, midy + 23);
    arc(30 + i, midy + 23, 0, 180, 12);
    line(42 + i, midy + 23, 78 + i, midy + 23);
    arc(90 + i, midy + 23, 0, 180, 12);
    line(102 + i, midy + 23, 120 + i, midy + 23);
    line(28 + i, midy, 43 + i, midy - 15);
    line(43 + i, midy - 15, 57 + i, midy - 15);
    line(57 + i, midy - 15, 57 + i, midy);
    line(57 + i, midy, 28 + i, midy);
    line(62 + i, midy - 15, 77 + i, midy - 15);
    line(77 + i, midy - 15, 92 + i, midy);
    line(92 + i, midy, 62 + i, midy);
    line(62 + i, midy, 62 + i, midy - 15);
    floodfill(5 + i, midy + 22, YELLOW);
    setcolor(BLUE);
```

```
    setfillstyle(SOLID_FILL, DARKGRAY);
    /*  Draw Wheels */
    circle(30 + i, midy + 25, 9);
    circle(90 + i, midy + 25, 9);
    floodfill(30 + i, midy + 25, BLUE);
    floodfill(90 + i, midy + 25, BLUE);
    /* Add delay of 0.1 milli seconds */
    delay(100);
  }

  getch();
  closegraph();
}
```

Output



□□□