Hardware and Software Requirements

For implementing graphics concepts in C-Language:

1. **Software Requirement:**

   Turbo C / C++ compiler that supports graphics.h package.

   special DOSBOXed installer for Turbo C++ compiler

   Download from following links:

   http://www.megaleecher.net/Download_Turbo_For_Windows

2. **Minimum hardware requirements:**

   Intel Pentium III800 MHz Processor or higher version

   Intel chipset 810 mother board or higher version

   14" color monitor or greater than that

   Mouse

   Keyboard

   2GB HDD or greater

   256 MB RAM or greater

For doing key frame animation:

Software Requirements: Adobe Flash Professional version 6 .

Download Adobe Flash CS6 which contains Flash Professional Also and install

Hardware Requirements:

Intel® Pentium® 4, Intel Centrino®, Intel Xeon®, or Intel Core™ Duo (or compatible)
processor

Microsoft® Windows® 7 (64 bit) or Windows 8 (64 bit)

4GB of RAM

2.5GB of available hard-disk space for installation; additional free space required during installation (cannot install on removable flash storage devices)

1024x768 display (1280x800 recommended)

QuickTime 10.x software recommended

BASIC GRAPHICS FUNCTION

1) INITGRAPH

- Initializes the graphics system.

Declaration

- Void far initgraph(int far *graphdriver)

Remarks

- To start the graphic system, you must first call initgraph.
- Initgraph initializes the graphic system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.
- Initgraph also resets all graphics settings (color, palette, current position, viewport, etc) to their defaults then resets graph.

2) GETPIXEL, PUTPIXEL

- Getpixel gets the color of a specified pixel.
- Putpixel places a pixel at a specified point.

Decleration

- 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 at (x, y).

Return value

- Getpixel returns the color of the given pixel.
- Putpixel does not return.

3) CLOSE GRAPH

- Shuts down the graphic system.

Decleration

- Void far closegraph(void);

Remarks

- Close graph deallocates all memory allocated by the graphic system.
- It then restores the screen to the mode it was in before you called initgraph.

Return value

- None.

4) ARC, CIRCLE, PIESLICE

- arc draws a circular arc.
- Circle draws a circle
- Pieslice draws and fills a circular pieslice

Decleration

- 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 pieslice in the current drawing color, then fills it using the current fill pattern and fill color.

5) ELLIPSE, FILLELIPSE, SECTOR

- Ellipse draws an elliptical arc.
- Fillellipse draws and fills ellipse.
- Sector draws and fills an elliptical pie slice.

Decleration

- 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 farsectoe(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 elliptical arc in the current drawing color and than fills it with fill color and fill pattern.
- Sector draws an elliptical pie slice in the current drawing color and than fills it using the pattern and color defined by setfillstyle or setfillpattern.

6) FLOODFILL

- Flood-fills a bounded region.

Decleration

- Void far floodfill(int x, int y, int border)

Remarks

- Floodfills an enclosed area on bitmap device.
- The area bounded by the color border is flooded with the current fill pattern and fill color.
- (x,y) is a "seed point"
    - ¾ If the seed is within an enclosed area, the inside will be filled.
    - ¾ If the seed is outside the enclosed area, the exterior will be filled.
- Use fillpoly instead of floodfill wherever possible so you can maintain code compatibility with future versions.
- Floodfill doesnot work with the IBM-8514 driver.

Return value

- If an error occurs while flooding a region, graph result returns '1'.

7) GETCOLOR, SETCOLOR

- Getcolor returns the current drawing color.
- Setcolor returns the current drawing color.

Decleration

- 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 set a drawing color with setcolor , you can pass either the color number or the equivalent color name.

8) LINE,LINEREL,LINETO

- Line draws a line between two specified pints.

- Onerel draws a line relative distance from current position(CP).
- Linrto draws a line from the current position (CP) to(x,y).

- 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 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).

Return value

- None

9) RECTANGLE

- Draws a rectangle in graphics mode.

Decleration

- Void far rectangle(int left, int top, int right, int bottom)

Remarks

- It 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.

Return value

- None.

ALGORITHM TO DRAW A LINE USING DDA ALGORITHM.

1. Start.
2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare

   gdriver=DETECT,gmode.
3. Initialise the graphic mode with the path location in TC folder.
4. Input the two line end-points and store the left end-points in (x1,y1).
5. Load (x1,y1) into the frame buffer;that is,plot the first point.put x=x1,y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1.
7. If abs(dx) > abs(dy), do s=abs(dx).
8. Otherwise s= abs(dy).
9. Then xi=dx/s and yi=dy/s.
10. Start from k=0 and continuing till k<s,the points will be i.

    x=x+xi.

    ii. y=y+yi.
11. Place pixels using putpixel at points (x,y) in specified colour.
12. Close Graph.
13. Stop.

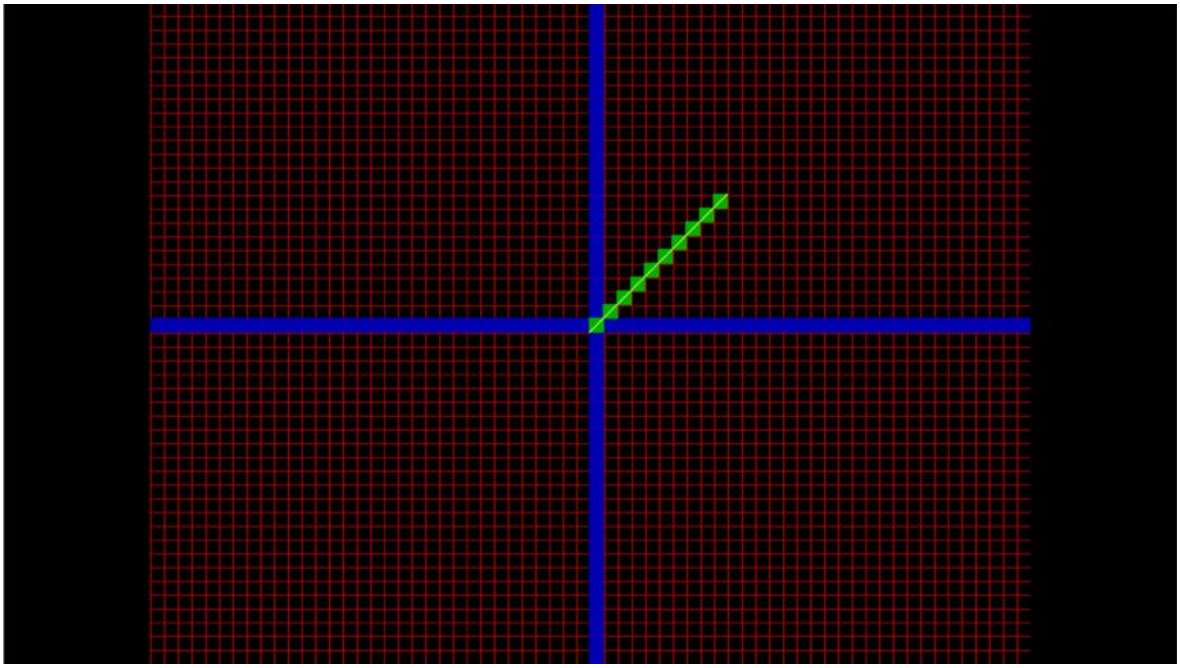
DDA CODE

```
// DDA Line Drawing Algorithm Using C Programming

#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<graphics.h>
#define round(val) (int)(val+0.5)
void grid(int maxx,int maxy,int size) // Drawing Grid
{
    int i,j;
    setcolor(RED);
    for(i=0;i<=maxx;i+=size)
```

```
            line(i,0,i,maxy); // Drawing Horizontal Lines
    for(i=0;i<=maxy;i+=size)
            line(0,i,maxx,i); // Drawing Vertical Lines
    for(i=0;i<=maxx;i+=size)
    {
            setfillstyle(SOLID_FILL,BLUE);
            bar(i,(maxy/2-size),i+size,maxy/2);
    } // Highlighting X-axis
    for(i=0;i<=maxy+1;i+=size)
    {
            setfillstyle(SOLID_FILL,BLUE);
            bar((maxx/2),i-size,maxx/2+size,i);
    } // Highlighting Y-axis
}

void plot(int x1,int y1,int size) // Plotting bars for highlighting pixel
{
    int x0=getmaxx()/2,y0=getmaxy()/2;
    int x=x0+(int)(size*x1);
    int y=y0-(int)(size*y1);
    setfillstyle(SOLID_FILL,GREEN);
    bar(x,y-size,x+size,y);
}

void drawline(int x1,int y1,int x2,int y2,int size) // Drawing Line through the pixel
{
    int x0=getmaxx()/2,y0=getmaxy()/2;
    int x11=x0+(int)(size*x1);
    int y11=y0-(int)(size*y1);
    int x22=x0+(int)(size*x2);
    int y22=y0-(int)(size*y2);

    setcolor(YELLOW);
    line(x11,y11,x22,y22);
}
void line_dda(int,int,int,int);

void main()
{
    int gd=DETECT,gm;
    int maxx,maxy,size=10;
```

```
    int xa,xb,ya,yb;
    printf("Enter the two values");
    scanf("%d%d%d%d",&xa,&ya,&xb,&yb);
    initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
    cleardevice();
    maxx=getmaxx();
    maxy=getmaxy();
    grid(maxx,maxy,size);
    line_dda(xa,ya,xb,yb);
    drawline(xa,ya,xb,yb,size);
    getch();
    closegraph();
}

void line_dda(int xa,int ya,int xb,int yb) // For implementing DDA
{
    int Dx=xb-xa,Dy=yb-ya,steps,k;
    float xin,yin,X=xa,Y=ya;
    if(abs(Dx)>abs(Dy))
       steps=abs(Dx);
    else
       steps=abs(Dy);
    setcolor(GREEN);
    xin=Dx/(float)steps;
    yin=Dy/(float)steps;

    for(k=0;k<steps;k++)
    {
            plot(floor(X),floor(Y),10);
            X=X+xin;
            Y=Y+yin;
    }
}
```

DDA OUTPUT



BRESENHAM'S ALGORITHM FOR LINE DRAWING.

1. Start.
2. Declare variables x,y,x1,y1,x2,y2,p,dx,dy and also declare gdriver=DETECT,gmode.
3. Initialize the graphic mode with the path location in TC folder.
4. Input the two line end-points and store the left end-points in (x1,y1).
5. Load (x1,y1) into the frame buffer; that is, plot the first point put x=x1,y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1,and obtain the initial value of decision parameter

   p as:

   $$p=(2dy-dx).$$
7. Starting from first point (x,y) perform the following test:
8. Repeat step 9 while(x<=x2).
9. If p<0, next point is (x+1,y) and p=(p+2dy).
10. Otherwise, the next point to plot is (x+1,y+1) and p=(p+2dy-2dx).
11. Place pixels using putpixel at points (x,y) in specified colour.
12. Close Graph.
13. Stop.

BRESENHAM'S Line Code

```
#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

#define MAX 20

#define SIZE  10


inline void utpixel(int x,int y, int col) {

   floodfill(x*SIZE+321,239-y*SIZE,col);
```

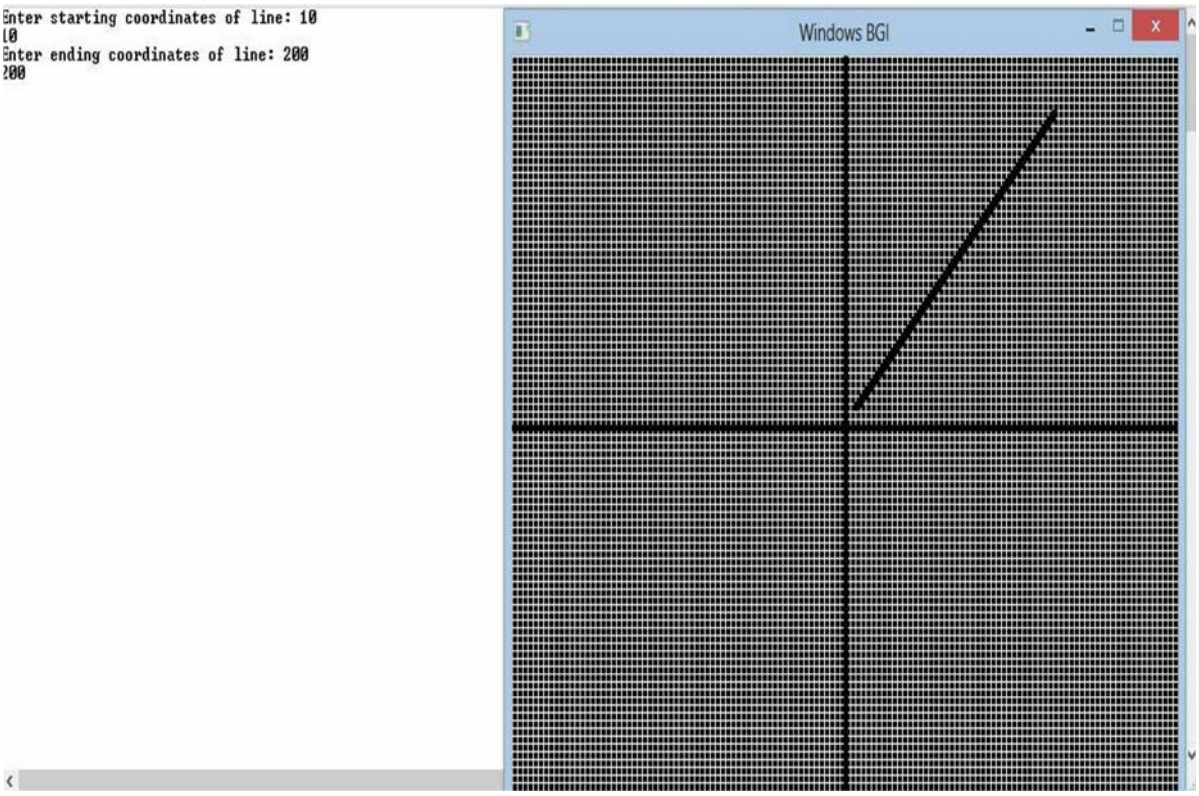
```
}

void setup() {
    for(int y=0; y< 480; y+= SIZE) {
        line(0,y,640,y);
    }

    for(int x = 0; x < 640 ; x+=SIZE) {
        line(x,0,x,480);
        floodfill(x-1,239,1);
    }

    for(int y=0; y< 480; y+= SIZE) {
        floodfill(321,y+1,14);
    }

}

void swap(int* a,int* b) {
    int t=*a;
    *a=*b;
    *b=t;
}
```

```
void midpointline(int x1,int y1,int x2,int y2) {
    int dx,dy,d,incry,incre,incrne,slopegt1=0;
    dx=abs(x1-x2);
    dy=abs(y1-y2);
    if(dy>dx) {
        swap(&x1,&y1);
        swap(&x2,&y2);
        swap(&dx,&dy);
        slopegt1=1;
    }
    if(x1>x2) {
        swap(&x1,&x2);
        swap(&y1,&y2);
    }
    if(y1>y2)
        incry=-1;
    else
        incry=1;
    d=2*dy-dx;
    incre=2*dy;
    incrne=2*(dy-dx);
    while(x1<x2) {
        if(d<=0)
            d+=incre;
```

```
        else {
            d+=incrne;
            y1+=incry;
        }
        x1++;
        if(slopegt1) {
            utpixel(y1,480-x1,1);
            //floodfill(y1*SIZE+321,239-(480-x1)*SIZE,1);
        } else {
            utpixel(x1,480-y1,1);
            //floodfill(x1*SIZE+321,239-(480-y1)*SIZE,1);
        }
    }
}
int main() {
    int n,i;
    int pt[MAX][2];
    int gd=DETECT,gm;
    printf("range of pixel (%d,%d) to (%d,%d)\n",-640/SIZE,-480/SIZE,640/SIZE,480/SIZE);
    printf("Enter the number of points:");
    scanf("%d",&n);
    printf("Enter the x and y coordinates:");
    for(i=0; i<n; i++) {
        scanf("%d %d",&pt[i][0],&pt[i][1]);
```

```
        pt[i][1]=480 - pt[i][1];
    }
    initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
    setup();
    for(i=0; i<n-1; i++) {
        midpointline(pt[i][0],pt[i][1],pt[i+1][0],pt[i+1][1]);
        //outtextxy(pt[i+1][0]-2,pt[i+1][1]-3,"*");
    }
    getch();
    closegraph();
}
```

BRESENHAM'S Line OUTPUT



BRESENHAM'S ALGORITHM TO DRAW A CIRCLE.

1. Start.
2. Declare variables x,y,p and also declare gdriver=DETECT,gmode.
3. Initialise the graphic mode with the path location in TC folder.
4. Input the radius of the circle r.
5. Load x=0,y=r,initial decision parameter p=1-r.so the first point is (0,r).
6. Repeat Step 7 while (x<y) and increment x-value simultaneously.
7. If (p>0),do p=p+2*(x-y)+1.
8. Otherwise p=p+2*x+1 and y is decremented simultaneously.
9. Then calculate the value of the function circlepoints() with p.arameters (x,y).
10. Place pixels using putpixel at points (x+300,y+300) in specified colour in circlepoints()

    function shifting the origin to 300 on both x-axis and y-axis.

11. Close Graph.
12. Stop.

BRESENHAM'S CIRCLE CODE

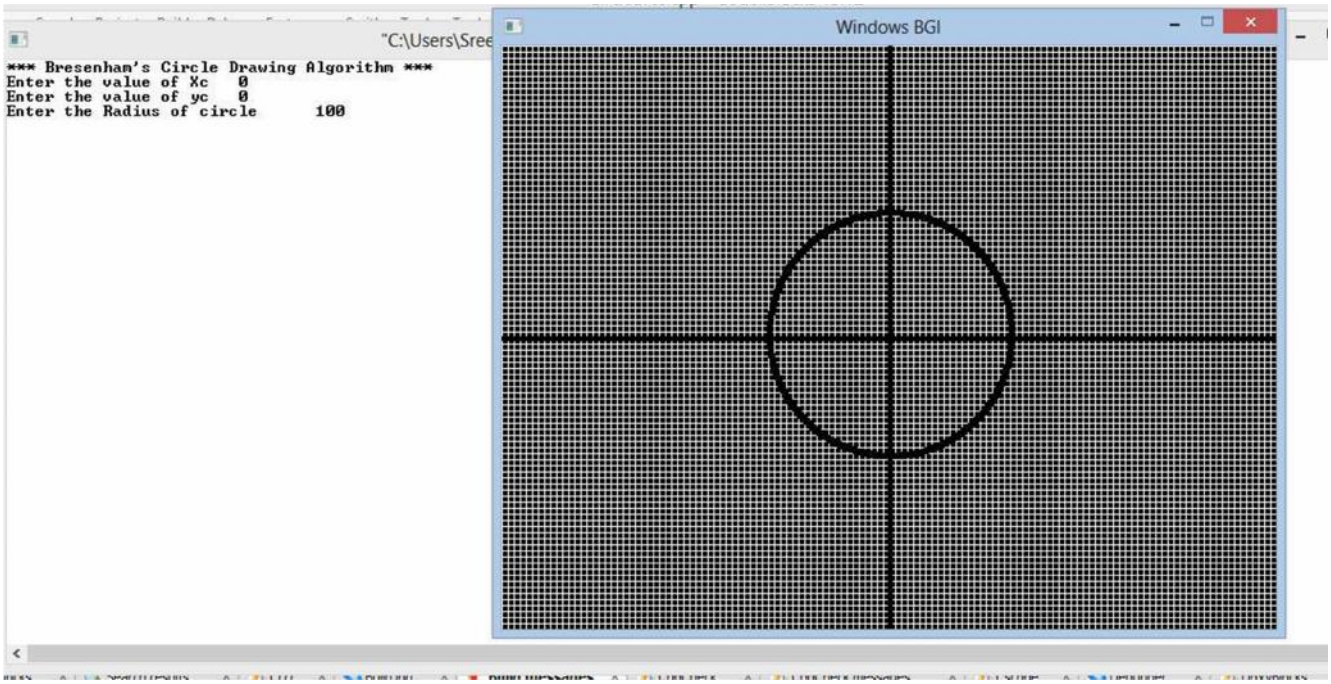
```
#include<graphics.h>

#include<stdio.h>

#define SIZE  10


inline void utpixel(int x,int y, int col){

   floodfill(x*SIZE+321,239-y*SIZE,col);

}
```

```
void setup() {
    for(int y=0; y< 480; y+= SIZE) {
        line(0,y,640,y);
    }

    for(int x = 0; x < 640 ; x+=SIZE) {
        line(x,0,x,480);
        floodfill(x-1,239,1);
    }

    for(int y=0; y< 480; y+= SIZE) {
        floodfill(321,y+1,14);
    }

}

int main() {
    int gd=DETECT,gm;
    int d,r,x,y,xc,yc;



    printf("Enter Radius\n");
    scanf("%d",&r);
    printf("Enter Center of circle\n");
```

```
scanf("%d",&xc);
scanf("%d",&yc);
initgraph(&gd,&gm,NULL);
setup();
d=3-2*r;
x=0;
y=r;

while(x<=y) {
    utpixel(xc+x,yc+y,5);
    utpixel(xc-y,yc-x,5);
    utpixel(xc+y,yc-x,5);
    utpixel(xc-y,yc+x,5);
    utpixel(xc+y,yc+x,5);
    utpixel(xc-x,yc-y,5);
    utpixel(xc+x,yc-y,5);
    utpixel(xc-x,yc+y,5);
    if(d<=0) {
        d=d+4*x+6;
    } else {
```

```
            d=d+4*x-4*y+10;
            y=y-1;
        }
        x=x+1;
    }
    getch();
}
```

BRESENHAM'S CIRCLE OUTPUT



WRITE A PROGRAM TO DRAW AN ELLIPSE USING MID-POINT LINE DRAWING ALGORITHM

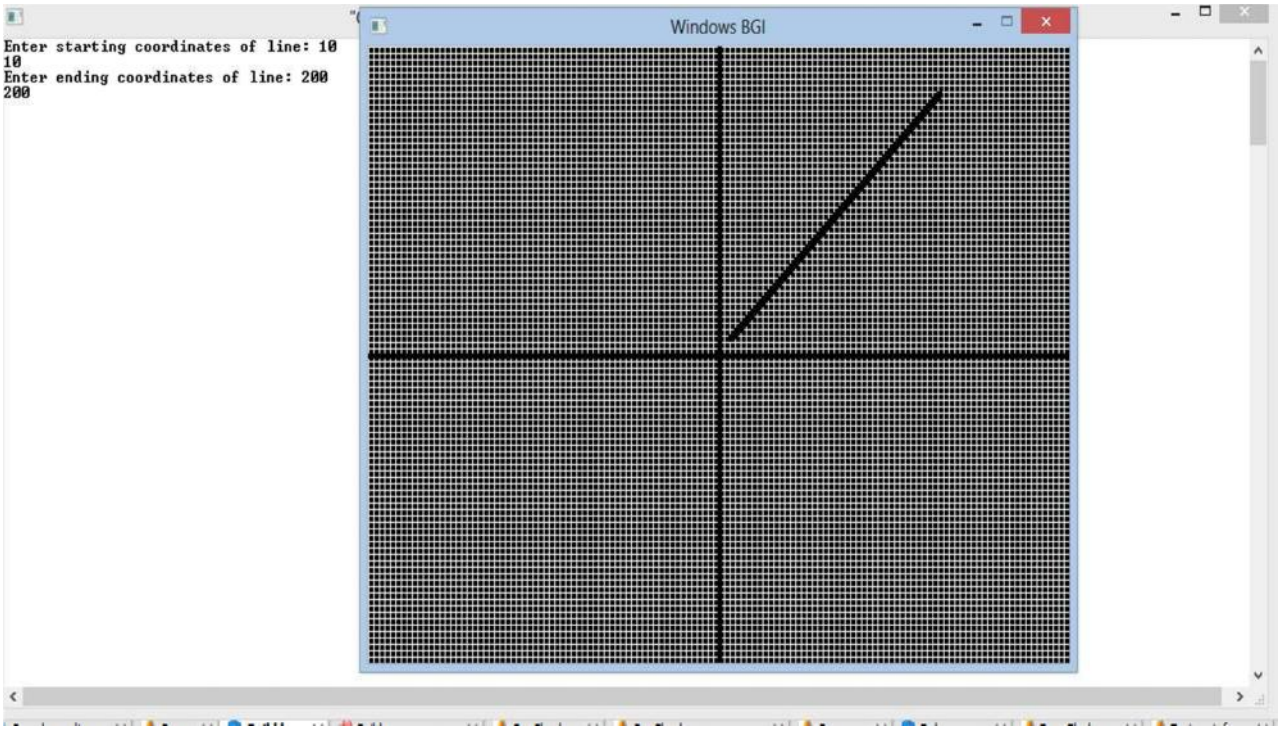
```
#include<stdio.h>
#include<graphics.h>
#include<stdlib.h>
#define MAX 20
#define SIZE  10
inline void utpixel(int x,int y, int col) {
   floodfill(x*SIZE+321,239-y*SIZE,col);
}
void setup() {
   for(int y=0; y< 480; y+= SIZE) {
      line(0,y,640,y);
   }
 for(int x = 0; x < 640 ; x+=SIZE) {
      line(x,0,x,480);
      floodfill(x-1,239,1);
   }
for(int y=0; y< 480; y+= SIZE) {
      floodfill(321,y+1,14);
   }
}
void swap(int* a,int* b) {
   int t=*a;
```

```
    *a=*b;
    *b=t;
}
void midpointline(int x1,int y1,int x2,int y2) {
    int dx,dy,d,incry,incre,incrne,slopegt1=0;
    dx=abs(x1-x2);
    dy=abs(y1-y2);
    if(dy>dx) {
        swap(&x1,&y1);
        swap(&x2,&y2);
        swap(&dx,&dy);
        slopegt1=1;
    }
    if(x1>x2) {
        swap(&x1,&x2);
        swap(&y1,&y2);
    }
    if(y1>y2)
        incry=-1;
    else
        incry=1;
    d=2*dy-dx;
    incre=2*dy;
    incrne=2*(dy-dx);
```

```
    while(x1<x2) {
        if(d<=0)
            d+=incre;
        else {
            d+=incrne;
            y1+=incry;
        }
        x1++;
        if(slopegt1) {
            utpixel(y1,480-x1,1);
            //floodfill(y1*SIZE+321,239-(480-x1)*SIZE,1);
        } else {
            utpixel(x1,480-y1,1);
            //floodfill(x1*SIZE+321,239-(480-y1)*SIZE,1);
        }
    }
}
int main() {
    int n,i;
    int pt[MAX][2];
    int gd=DETECT,gm;
    printf("range of pixel (%d,%d) to (%d,%d)\n",-640/SIZE,-480/SIZE,640/SIZE,480/SIZE);

    printf("Enter the number of points:");
```

```
    scanf("%d",&n);

    printf("Enter the x and y coordinates:");

   for(i=0; i<n; i++) {
        scanf("%d %d",&pt[i][0],&pt[i][1]);
        pt[i][1]=480 - pt[i][1];
    }
    initgraph(&gd,&gm,NULL);
    setup();
    for(i=0; i<n-1; i++) {
        midpointline(pt[i][0],pt[i][1],pt[i+1][0],pt[i+1][1]);
        //outtextxy(pt[i+1][0]-2,pt[i+1][1]-3,"*");
    }
    getch();
    closegraph();
}
```

OUTPUT OF MIDPOINT LINE ALGORITHM



WRITE A PROGRAM TO DRAW A CIRCLE USING MID-POINT CIRCLE DRAWING ALGORITHM.

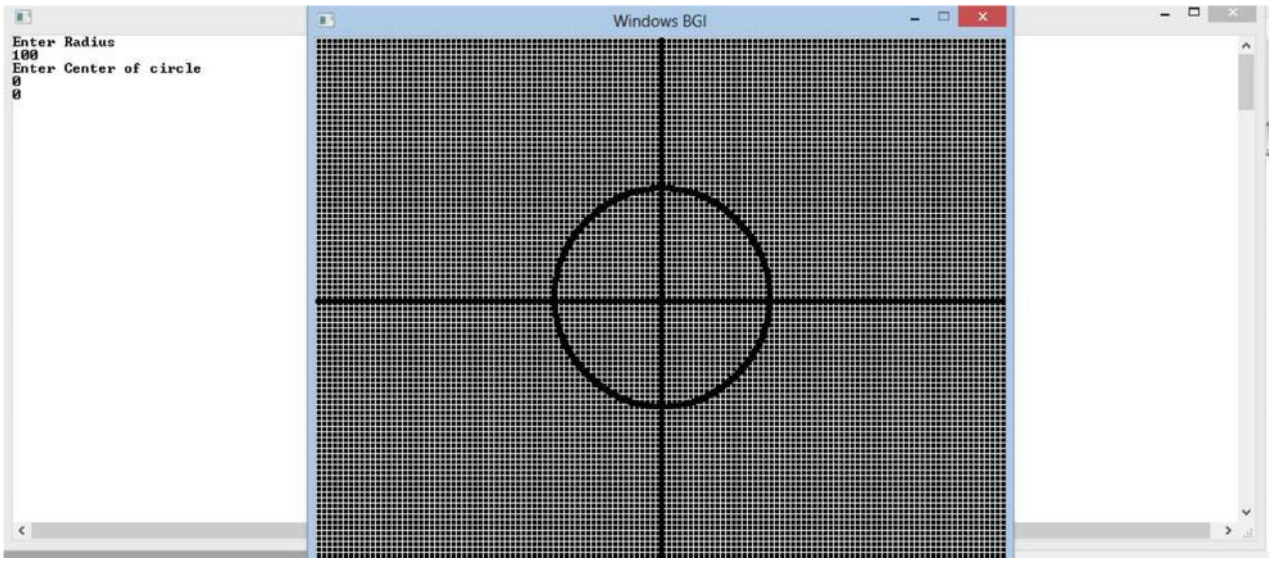
```
#include<graphics.h>

#include<stdio.h>

#define SIZE  10


inline void utpixel(int x,int y, int col) {

   floodfill(x*SIZE+321,239-y*SIZE,col);

}
```

```
void setup() {
    for(int y=0; y< 480; y+= SIZE) {
        line(0,y,640,y);
    }
    for(int x = 0; x < 640 ; x+=SIZE) {
        line(x,0,x,480);
        floodfill(x-1,239,1);
    }
    for(int y=0; y< 480; y+= SIZE) {
        floodfill(321,y+1,14);
    }
}

int main() {
    int gd=DETECT,gm;
    int i,r,x,y,xc,yc;
    float d;



    printf("Enter Radius\n");
    scanf("%d",&r);
    printf("Enter Center of circle\n");
    scanf("%d",&xc);
    scanf("%d",&yc);
```

```
d=1.25-r;
x=0;
y=r;
initgraph(&gd,&gm,NULL);
setup();

do {
   if(d<0) {
      x=x+1;
      d=d+2*x+1;
   } else {
      x=x+1;
      y=y-1;
      d=d+2*x-2*y+10;
   }
   utpixel(xc+x,yc+y,5);
   utpixel(xc-y,yc-x,5);
   utpixel(xc+y,yc-x,5);
   utpixel(xc-y,yc+x,5);
   utpixel(xc+y,yc+x,5);
   utpixel(xc-x,yc-y,5);
```

```
      utpixel(xc+x,yc-y,5);
       utpixel(xc-x,yc+y,5);
   } while(x<y);
   getch();
}
```

OUTPUT OF MID-POINT CIRCLE



WRITE A PROGRAM TO DRAW AN ELLIPSE USING MID-POINT ELLIPSE DRAWING ALGORITHM.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void ellips(int x,int y);
void completellipse(int r,int g,int u,int v)
{
        float s,k,e,f,x;
        double p1,p2;
        s=r;k=g;
        e=(pow((s+.5),2));
        f=(pow((k-1),2));
        p2=((u*e)+(v*f)-(u*v));
        ellips(s,k);
        while(k>=0)
         {
                if(p2>0)
                        p2=(p2+v-(2*v*s));
                else
                {
                        p2=(p2+(2*u*(s+1))-(2*v*(k-1))+v);
                        s++;
                }
               k--;
```

```
                ellips(s,k);
            }
    }
void main()
    {
        int gdriver=DETECT,gmode;
        int a,b,x,y; long u,v,p1;
        initgraph(&gdriver,&gmode,"C:\\tc\\bgi::")
        ; printf("\n enter the length of major axis:");
        scanf("\t%d",&a);
        printf("\n enter the length of minor axis:");
        scanf("\t%d",&b);
        x=0; y=b;
        u=pow(b,2);
        v=pow(a,2);
        p1=(u-(v*b)+(.25*v));
        ellips(x,y);
        while(2*(u*x)<=2*(v*y))
        {
            x++;
            if(p1<0)
                p1=(p1+(2*u*v)+v);
            else
            {
                p1=(p1+(2*u*x)-(2*v*y)+u);
                y--;
            }
```

```
                ellips(x,y);
        }
        completellipse(x,y,u,v);
        getch();
        closegraph();
}
void ellips(int x,int y)
{
        putpixel(x+200,y+200,8);
        putpixel(-x+200,y+200,8);
        putpixel(x+200,-y+200,8);
        putpixel(-x+200,-y+200,8);
}
```

OUTPUT OF MID-POINT ELLIPSE