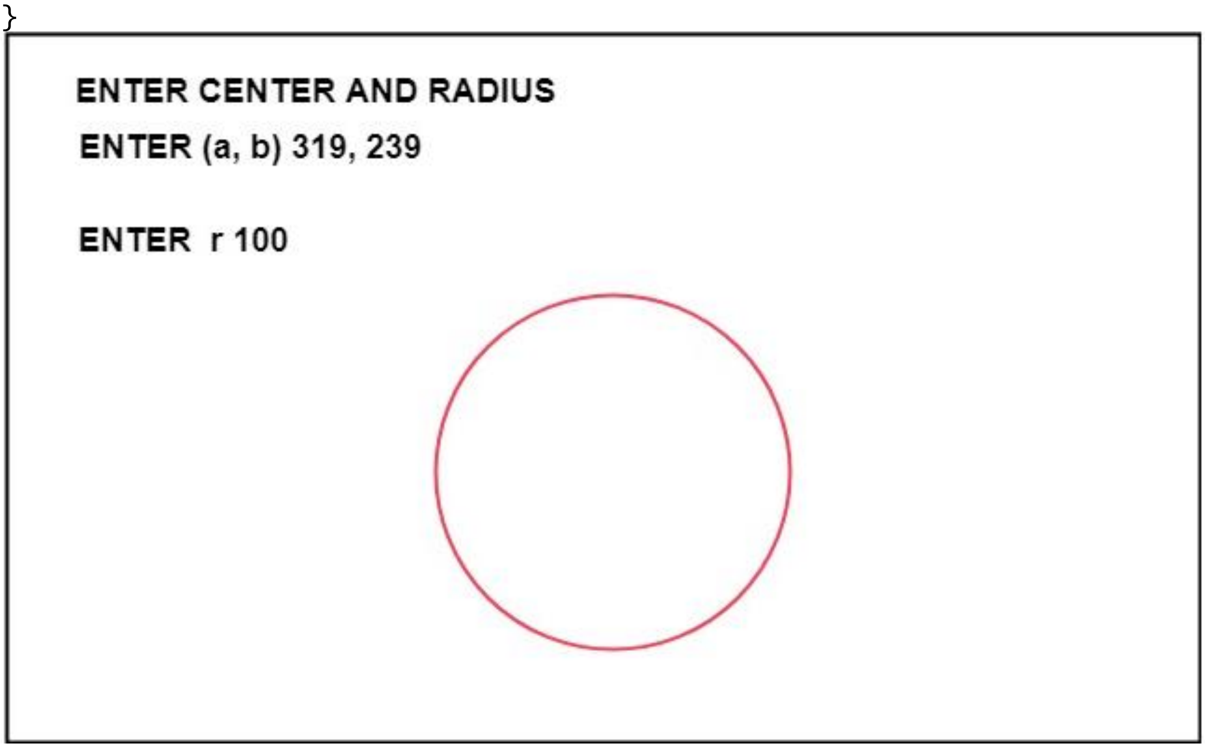
ARYAMAN MISHRA

19BCE1027

Midpoint circle drawing

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

class bresen
{
float x, y,a, b, r, p;
public:
void get ();
void cal ();
};
void main ()
{
bresen b;
b.get ();
b.cal ();
getch ();
}
Void bresen :: get ()
{
cout<<"ENTER CENTER AND RADIUS";
cout<< "ENTER (a, b)";
cin>>a>>b;
cout<<"ENTER r";      cin>>r;
}
void bresen ::cal ()
{
```

```
/* request auto detection */
int gdriver = DETECT,gmode, errorcode;
int midx, midy, i;
/* 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 */
}
x=0;
y=r;
putpixel (a, b+r, RED);
putpixel (a, b-r, RED);
putpixel (a-r, b, RED);
putpixel (a+r, b, RED);
p=5/4)-r;
while (x<=y)
{
If (p<0)
p+= (4*x)+6;
else
{
p+=(2*(x-y))+5;
y--;
}
x++;
putpixel (a+x, b+y, RED);
putpixel (a-x, b+y, RED);
putpixel (a+x, b-y, RED);
putpixel (a+x, b-y, RED);
putpixel (a+x, b+y, RED);
putpixel (a+x, b-y, RED);
putpixel (a-x, b+y, RED);
putpixel (a-x, b-y, RED);
}
}
```



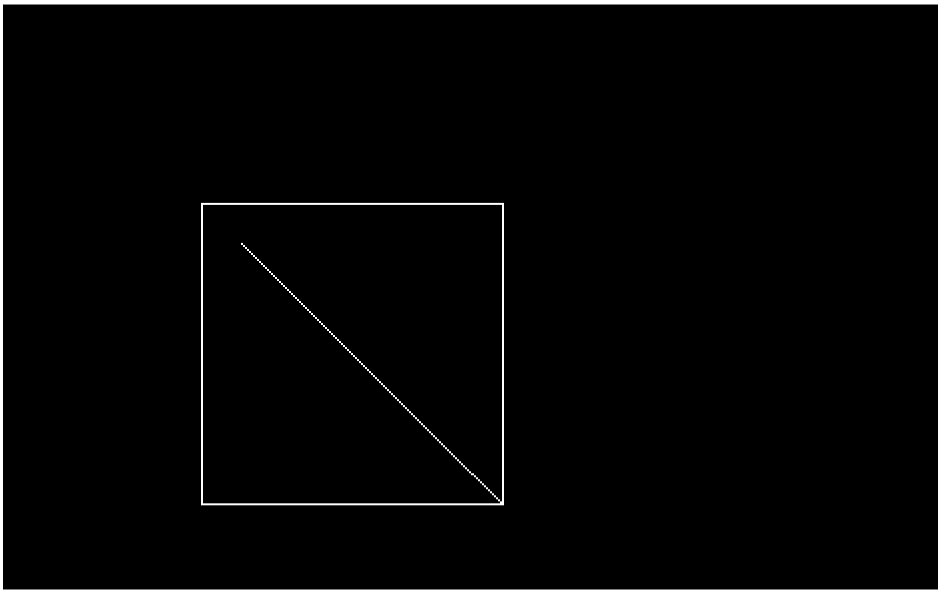
```
ENTER CENTER AND RADIUS
ENTER (a, b) 319, 239

ENTER  r 100
```

Liang- Barsky Line clipping

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

        x1=120;
        y1=120;
        x2=300;
        y2=300;

        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();
}
```



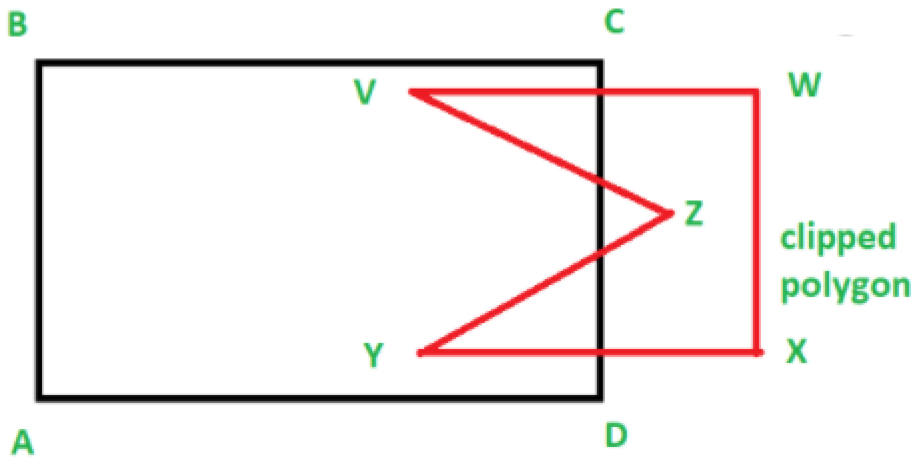
Weiler Atherton Polygon clipping

```
#include<iostream>
#include<graphics.h>
#include<utility>
using namespace std;
struct vertex
{
    float x;
    float y;
};
vertex cw[40], sp[40];
int n_cw, n_sp;
void draw_poly(vertex vlist[], int n)
{
    for (int i=0; i<n; i++)
        line(vlist[i].x, vlist[i].y, vlist[(i+1)%n].x, vlist[(i+1)%n].y);
}
int in_out(float x, float y, int x1, int y1, int x2, int y2)
{
    float p = (y-y1)*(x2-x1) - (x-x1)*(y2-y1);
    if (p<0)
        return 0;  //for out
    return 1;    //for in
}
void intersection_lineseg(float &x, float &y, int x1, int y1, int x2,
int y2, int xa, int ya, int xb, int yb)
{
    x = -1;
    y = -1;
    if (x2==x1 && xb==xa)
        return;
    else if (x2==x1)
    {
        float m2 = (yb-ya) / (float) (xb-xa);
        x = x1;
        y = ya - m2*(xa-x1);
    }
    else if (xb==xa)
    {
        float m1 = (y2-y1) / (float) (x2-x1);
        x = xa;
        y = y1 + m1*(xa-x1);
    }
    else
    {
        float m1 = (y2-y1) / (float) (x2-x1);
        float m2 = (yb-ya) / (float) (xb-xa);
        if (m1==m2)
            return;
        x = (ya-y1 + m1*x1 - m2*xa) / (m1-m2);
        y = (m1*m2*(xa-x1) + m2*y1 - m1*ya) / (m2-m1);
    }
    if ((x1>=x2 && (x<x2||x>x1)) || (x2>=x1 && (x>x2||x<x1)) ||
(y1>=y2 && (y<y2||y>y1)) || (y2>=y1 && (y>y2||y<y1))
            || (xa>=xb && (x<xb||x>xa)) || (xb>=xa &&
(x>xb||x<xa)) || (ya>=yb && (y<yb||y>ya)) || (yb>=ya &&
(y>yb||y<ya)))
    {
        x = -1;
        y = -1;
    }
}
void wa_clip()
{
    vertex tempcw[40], tempsp[40];
    int tag_sp[40], tag_cw[40], trav_sp[40], trav_cw[40];
    float x,y;
    int entry_list[10];      //saves indexes only
    int e=-1;
    //for new cw array
    int kc=-1;   //first vertex gets added last in array
    for (int i=0; i<n_cw; i++)
    {
        vertex tempi[20][2];           //for ordering intersection
points, 2nd column's x is for tag
        int ti = -1;
        for (int j=0; j<n_sp; j++)
        {
            intersection_lineseg(x, y, cw[i].x, cw[i].y,
cw[(i+1)%n_cw].x, cw[(i+1)%n_cw].y,
                                sp[j].x, sp[j].y, sp[(j+1)%n_sp].x,
sp[(j+1)%n_sp].y);
            if (x==-1)  //or y==-1
                continue;
            ti++;
            tempi[ti][0].x = x;
            tempi[ti][0].y = y;
            int p1 = in_out(sp[j].x, sp[j].y, cw[i].x, cw[i].y,
cw[(i+1)%n_cw].x, cw[(i+1)%n_cw].y);
            int p2 = in_out(sp[(j+1)%n_sp].x, sp[(j+1)%n_sp].y,
cw[i].x, cw[i].y, cw[(i+1)%n_cw].x, cw[(i+1)%n_cw].y);
            if (p1==1 && p2==0)
                tempi[ti][1].x = 1;
            else
                tempi[ti][1].x = 0;
        }
        if (ti!=-1)
        {
            if (cw[(i+1)%n_cw].x > cw[i].x)             //sort
intersection points
            {
                //increasing x sort
                int min_idx;
                for (int k=0; k<ti; k++)
                {
                    min_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].x < tempi[min_idx][0].x)
                            min_idx=m;
                    }
                    float temp = tempi[min_idx][0].x;
                    tempi[min_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[min_idx][0].y;
                    tempi[min_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[min_idx][1].x;
                    tempi[min_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else if (cw[(i+1)%n_cw].x < cw[i].x)
            {
                //decreasing x sort
                int max_idx;
                for (int k=0; k<ti; k++)
                {
                    max_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].x > tempi[max_idx][0].x)
                            max_idx=m;
                    }
                    float temp = tempi[max_idx][0].x;
                    tempi[max_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[max_idx][0].y;
                    tempi[max_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[max_idx][1].x;
                    tempi[max_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else if (cw[(i+1)%n_cw].y > cw[i].y)
            {
                //increasing y sort
                int min_idx;
                for (int k=0; k<ti; k++)
                {
                    min_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].y < tempi[min_idx][0].y)
                            min_idx=m;
                    }
                    float temp = tempi[min_idx][0].x;
                    tempi[min_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[min_idx][0].y;
                    tempi[min_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
```

```
                    temp = tempi[min_idx][1].x;
                    tempi[min_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else
            {
                //decreasing y sort
                int max_idx;
                for (int k=0; k<ti; k++)
                {
                    max_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].y > tempi[max_idx][0].y)
                            max_idx=m;
                    }
                    float temp = tempi[max_idx][0].x;
                    tempi[max_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[max_idx][0].y;
                    tempi[max_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[max_idx][1].x;
                    tempi[max_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            for (int k=0; k<=ti; k++)                              //put sorted
intersection points in cw array
            {
                kc++;
                tempcw[kc].x = tempi[k][0].x;
                tempcw[kc].y = tempi[k][0].y;
                tag_cw[kc] = tempi[k][1].x;
                trav_cw[kc] = 0;
            }
        }
        kc++;
        tempcw[kc].x = cw[(i+1)%n_cw].x;
        tempcw[kc].y = cw[(i+1)%n_cw].y;
        tag_cw[kc] = -1;
        trav_cw[kc] = 0;
    }
    //for new sp array
    int ks=-1;    //first vertex gets added last in array
    for (int i=0; i<n_sp; i++)
    {
        vertex tempi[20][2];           //for ordering intersection
points, 2nd column's x is for tag
        int ti = -1;
        for (int j=0; j<n_cw; j++)
        {
            intersection_lineseg(x, y, cw[j].x, cw[j].y,
cw[(j+1)%n_cw].x, cw[(j+1)%n_cw].y,
                                 sp[i].x, sp[i].y, sp[(i+1)%n_sp].x,
sp[(i+1)%n_sp].y);
            if (x==-1)  //or y==-1
                continue;
            ti++;
            tempi[ti][0].x = x;
            tempi[ti][0].y = y;
            int p1 = in_out(sp[i].x, sp[i].y, cw[j].x, cw[j].y,
cw[(j+1)%n_cw].x, cw[(j+1)%n_cw].y);
            int p2 = in_out(sp[(i+1)%n_sp].x, sp[(i+1)%n_sp].y,
cw[j].x, cw[j].y, cw[(j+1)%n_cw].x, cw[(j+1)%n_cw].y);
            if (p1==1 && p2==0) {
                tempi[ti][1].x = 0;
                }
            else {
                tempi[ti][1].x = 1;
            }
        }
        if (ti!=-1)
        {
            if (sp[(i+1)%n_sp].x > sp[i].x)            //sort intersection
points
            {
                //increasing x sort
                int min_idx;
                for (int k=0; k<ti; k++)
                {
                    min_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].x < tempi[min_idx][0].x)
                            min_idx=m;
                    }
                    float temp = tempi[min_idx][0].x;
                    tempi[min_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[min_idx][0].y;
                    tempi[min_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[min_idx][1].x;
                    tempi[min_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else if (sp[(i+1)%n_sp].x < sp[i].x)
            {
                //decreasing x sort
                int max_idx;
                for (int k=0; k<ti; k++)
                {
                    max_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].x > tempi[max_idx][0].x)
                            max_idx=m;
                    }
                    float temp = tempi[max_idx][0].x;
                    tempi[max_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[max_idx][0].y;
                    tempi[max_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[max_idx][1].x;
                    tempi[max_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else if (sp[(i+1)%n_sp].y > sp[i].y)
            {
                //increasing y sort
                int min_idx;
                for (int k=0; k<ti; k++)
                {
                    min_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].y < tempi[min_idx][0].y)
                            min_idx=m;
                    }
                    float temp = tempi[min_idx][0].x;
                    tempi[min_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[min_idx][0].y;
```

```
                    tempi[min_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[min_idx][1].x;
                    tempi[min_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            else
            {
                //decreasing y sort
                int max_idx;
                for (int k=0; k<ti; k++)
                {
                    max_idx = k;
                    for (int m=k+1; m<ti+1; m++)
                    {
                        if (tempi[m][0].y > tempi[max_idx][0].y)
                            max_idx=m;
                    }
                    float temp = tempi[max_idx][0].x;
                    tempi[max_idx][0].x = tempi[k][0].x;
                    tempi[k][0].x = temp;
                    temp = tempi[max_idx][0].y;
                    tempi[max_idx][0].y = tempi[k][0].y;
                    tempi[k][0].y = temp;
                    temp = tempi[max_idx][1].x;
                    tempi[max_idx][1].x = tempi[k][1].x;
                    tempi[k][1].x = temp;
                }
            }
            for (int k=0; k<=ti; k++)                              //put sorted
intersection points in sp array
            {
                ks++;
                tempsp[ks].x = tempi[k][0].x;
                tempsp[ks].y = tempi[k][0].y;
                tag_sp[ks] = tempi[k][1].x;
                if (tag_sp[ks]==1) {
                    e++;
                    entry_list[e] = ks;
                }
                trav_sp[ks] = 0;
            }
        }
        ks++;
        tempsp[ks].x = sp[(i+1)%n_sp].x;
        tempsp[ks].y = sp[(i+1)%n_sp].y;
        tag_sp[ks] = -1;
        trav_sp[ks] = 0;
    }
    n_cw = kc+1;
    n_sp = ks+1;
    //traversal
    for (int i=0; i<=e; i++)
    {
        bool done = false;
        int j = entry_list[i];
        while(!done)
        {
            if (trav_sp[j] == 1)
                done = true;
            else if (tag_sp[j] == 1 || tag_sp[j] == -1)
            {
                line(tempsp[j].x, tempsp[j].y, tempsp[(j+1)%n_sp].x,
tempsp[(j+1)%n_sp].y);
                trav_sp[j] = 1;
                j++;
            }
            else if (tag_sp[j] == 0)
            {
                trav_sp[j] = 1;
                //swap
                for (int k=0; k<n_cw; k++)     //find location to switch
to
                {
                    if (tempcw[k].x==tempsp[j].x &&
tempcw[k].y==tempsp[j].y)
                    {
                        j = k;
                        break;
                    }
                }
                swap(tempcw, tempsp);
                swap(tag_cw, tag_sp);
                swap(trav_cw, trav_sp);
                int n = n_cw;
                n_cw = n_sp;
                n_sp = n_cw;
            }
        }
    }
}
```

```
int main()
{
    int gd=DETECT, gm=0;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    cout<<"Enter no. of vertices in clipping window"<<endl;
    cin>>n_cw;
    cout<<"Enter vertices (x,y) clockwise"<<endl;
    for (int i=0; i<n_cw; i++)
        cin>>cw[i].x>>cw[i].y;
    draw_poly(cw, n_cw);
    cout<<"Enter no. of vertices in subject polygon"<<endl;
    cin>>n_sp;
    cout<<"Enter vertices (x,y) clockwise"<<endl;
    for (int i=0; i<n_sp; i++)
        cin>>sp[i].x>>sp[i].y;
    draw_poly(sp, n_sp);
    char ch;
    cout<<"Press a key to clip"<<endl;
    cin>>ch;
    cleardevice();
    draw_poly(cw, n_cw);
    wa_clip();
    getch();
    closegraph();
    return 0;
}
```


B
C
V
W
Z
clipped
polygon
Y
X
A
D