```
void MidPointLine(int x0, int y0, int x1, int y1, int color) {
    int dx = abs(x1 - x0); int sx = x0 < x1? 1 : -1; //每次在x方向上+1或-1
    int dy = abs(y1 - y0); int sy = y0 < y1? 1 : -1; //每次在y方向上+1或-1
    int a = -sy * dy, b = sx * dx;
    //若斜率为正
    if (sx * sy >= 0) {
        //每次在x方向上+1或-1
        if (abs(a / b) <= 1) {//0<=斜率<= 1
            int delta1 = 2 * (a + b);
            int delta2 = 2 * a;
            int d = 2 * a + b; //d的初值d0
            int x = x0, y = y0;
            putpixel(x, y, color);

            while (sx * (x1 - x) >= 0) {//到终点为止
                if (d * sx <= 0) {//d＜0，则(x,y)更新为(x+1,y+1)，d更新为：d=d+2a+2b
                    x = x + sx;
                    y = y + sy;
                    d += delta1;
                }
                else if (d * sx > 0) {//d≥0，则(x,y)更新为(x+1,y)，d更新为：d=d+2a
                    x = x + sx;
                    d += delta2;
                }
                putpixel(x, y, color);
                cout << x << " " << y << endl;
            }
        }
```

计算初始值a=y0-y1，b=x1-x0，d=2a+b，x=x0，y=y0

绘制点(x,y)，判断d的符号：

若d<0，则(x,y)更新为(x+1,y+1)，d更新为：d+2a+2b

否则(x,y)更新为(x+1,y)，d更新为：d+2a

```
        //每次在y方向上+1或-1
        else {//斜率>1
            //用-a替换b,用-b替换a
            int delta1 = -2 * (a + b);
            int delta2 = -2 * b;
            int d = -2 * b - a;
            int x = x0, y = y0;
            putpixel(x, y, color);
            while (sy * (y1 - y) > 0) {
                if (d * sx <= 0) {
                    y = y + sy;
                    x = x + sx;
                    d += delta1;}
                else {y = y + sy;
                    d += delta2;}
                putpixel(x, y, color); }}}
    else { //斜率为负
        //每次在x方向上+1或-1
        if (abs(a / b) <= 1) { //-1<=斜率<= 0
            //注意这里是-b（增量改变）
            int delta1 = 2 * (a - b);
            int delta2 = 2 * a;
            int d = 2 * a - b;
            int x = x0, y = y0;
            putpixel(x, y, color);
            while (sx * (x1 - x) >= 0) {
                if (d * sx > 0) {
                    x = x + sx;
                    y = y + sy;
                    d += delta1;}
                else if (d * sx <= 0) {
                    x = x + sx;
                    d += delta2;}
                putpixel(x, y, color);
                cout << x << " " << y << endl;}}
        //每次在y上+1或-1
        else {//斜率< -1
            int delta1 = -2 * (-a + b); //用-a替换b,用-b替换a
            int delta2 = -2 * b;
            int d = -2 * b + a;
            int x = x0, y = y0;
            putpixel(x, y, color);
            while (sy * (y1 - y) > 0) {
                if (d * sx <= 0) {
                    y = y + sy;
                    x = x + sx;
                    d += delta1;}
                else {y = y + sy;
                    d += delta2;}
                putpixel(x, y, color);
                cout << x << " " << y << endl;}}}}
```

```
void BresenhamLine(int x0, int y0, int x1, int y1, int color)
{
    int dx = abs(x1 - x0); int sx = x0 < x1 ? 1 : -1; //起点终点位置关系
    int dy = abs(y1 - y0); int sy = y0 < y1 ? 1 : -1;
    int x, y, i; int e; float k;
    //如果斜率小于一，每次在x方向上+1或-1
        if (dy / dx < 1) {
            e = -dx; x = x0; y = y0;
            for (i = 0; i <= dx; i++) {
                putpixel(x, y, color);
                x = x + sx;
                //可能存在终点y值小于起点y值的情况，所以要利用sx和sy
                e += 2*(sy*dy);
                if (e * sy >= 0) {
                    y = y + sy;
                    e -= 2 * sy * dx;}}}
        else {e = -dy; //如果斜率大于1，每次在y方向上+1或-1
            x = x0;
            y = y0;
            for (i = 0; i    <= dy; i++) {
                putpixel(x, y, color);
                y = y + sy;
                e += 2 * sx * dx;
                if (e * sx >= 0) {
                    x = x + sx;
                    e -= 2 * sx * dy;}}}}
```

计算初始值△x，△y，e=-△x，$x=x_0$，$y=y_0$

绘制点(x,y)，e更新为e+2△y，判断e的符号：

若e＞0，则(x,y)更新为(x+1,y+1)，e更新为e-2△x

否则(x,y)更新为(x+1,y)

当直线没有画完时，重复步骤3。否则结束。

```
void putpix(int x, int y, int z, int color, int x1, int y1, int x2, int y2)
{
    putpixel(x, y, color);//原直线
    float k = float(x2 - x1) / (float)(y1 - y2);//-1/斜率
    double m;
    if (k < -1 || k>1)//垂直方向
    {
        for (int i = 1; i <= z; i++)
        {
            m = (k * (x - x1) + y1 - y - i) * (k * (x - x2) + y2 - y - i);
            if (m < 0)   //在两垂线之间
                putpixel(x, y + i, color);
            m = (k * (x - x1) + y1 - y + i) * (k * (x - x2) + y2 - y + i);
            if (m < 0)
                putpixel(x, y - i, color);
        }
        double l = fabs(z * k / sqrt(1 + k * k));
        setfillcolor(color);
        setlinecolor(color);
        fillcircle(x1, y1, l);
        fillcircle(x2, y2, l);
    }
    else //水平方向
    {
        for (int i = 1; i <= z; i++)
        {
            m = (k * (x + i - x1) + y1 - y) * (k * (x + i - x2) + y2 - y);
            if (m < 0)
                putpixel(x + i, y, color);
            m = (k * (x - i - x1) + y1 - y1 + 1) * (k * (x - i - x2) + y2 - y);
            if (m < 0)
                putpixel(x - i, y, color);
        }
        double b = z * sqrt(k * k + 1) / (k * k + 1);
        setfillcolor(color);
        setlinecolor(color);
        fillcircle(x1, y1, b);
        fillcircle(x2, y2, b);
    }
```

模块 2 裁剪算法

4. 实验题2-4　用逐边裁剪法实现多边形裁剪

注：代码最上方功能区注明是否处理退化边

5. 实验题2-5　用Weiler-Atherton裁剪法实现多边形裁剪（选做）