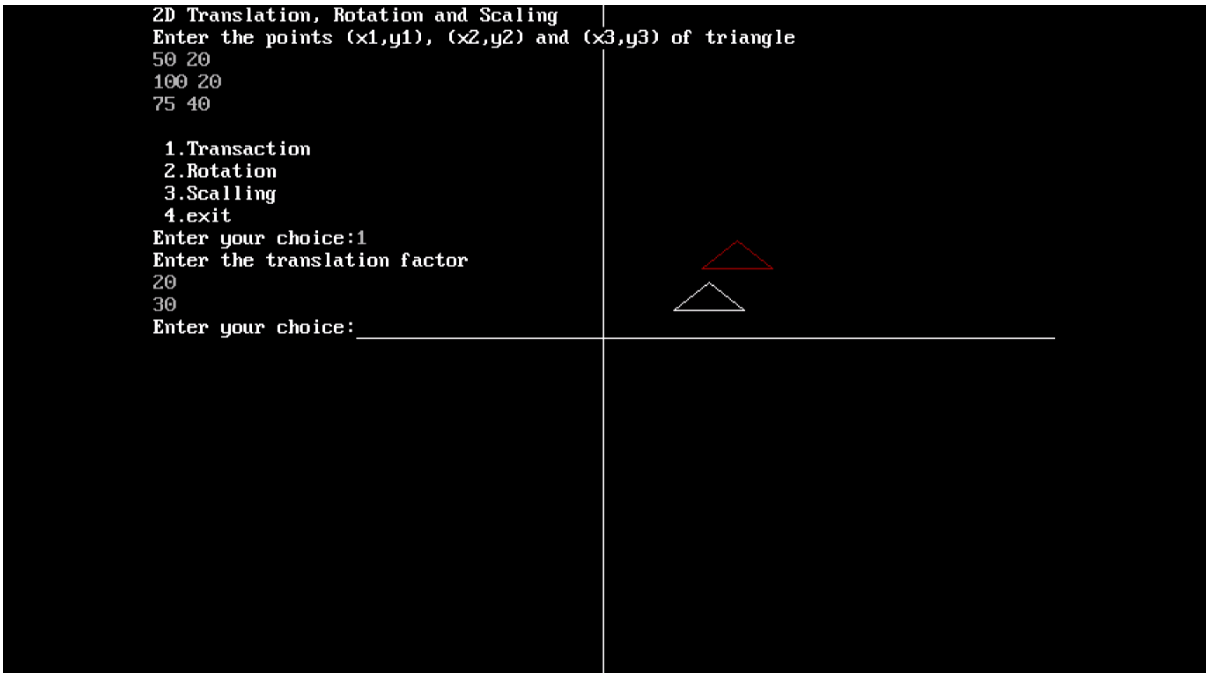
6. Write a Program for translation and scaling of the triangle

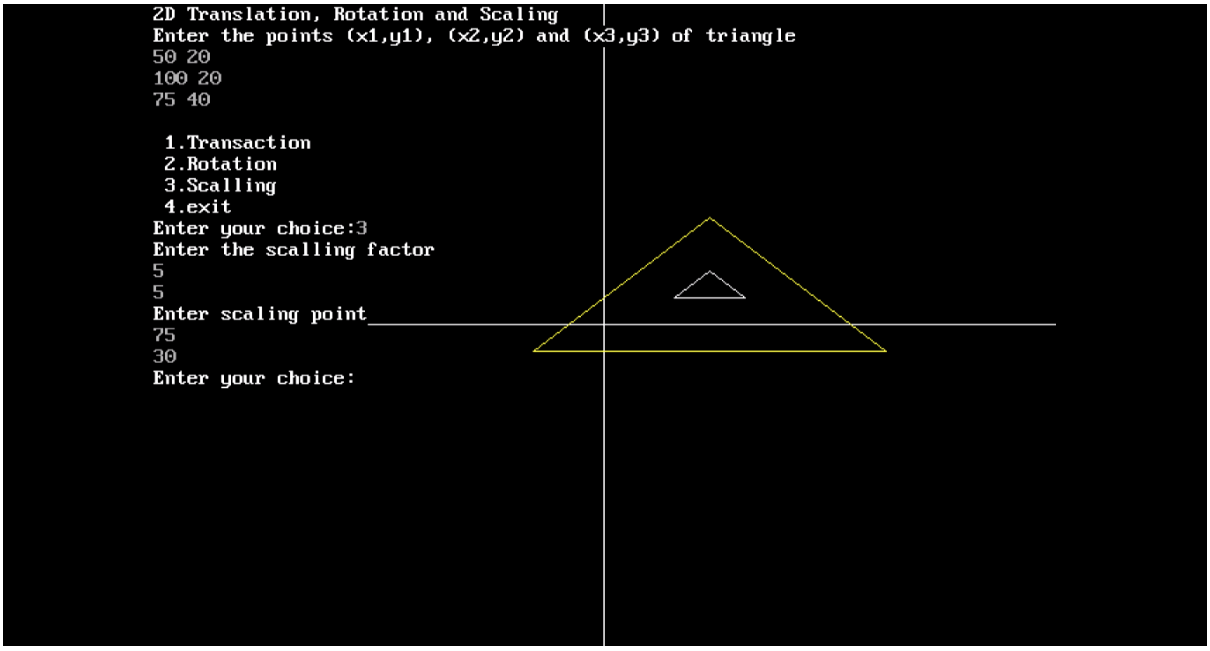
```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>

void main() {
   int gm, gd = DETECT, midx, midy, c;
   float x1, x2, x3, y1, y2, y3, x11, x22, x33, y11, y22, y33, sfx, sfy, tpx, tpy;

   initgraph(&gd, &gm, "..\\bgi");
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   line(midx, 0, midx, getmaxy());
   line(0, midy, getmaxx(), midy);

   printf("2D Translation and Scaling of a Triangle\n");
   printf("Enter the points (x1, y1), (x2, y2) and (x3, y3) of the triangle:\n");
   scanf("%f%f%f%f%f%f", &x1, &y1, &x2, &y2, &x3, &y3);

   setcolor(WHITE);
   line(midx + x1, midy - y1, midx + x2, midy - y2);
   line(midx + x2, midy - y2, midx + x3, midy - y3);
   line(midx + x3, midy - y3, midx + x1, midy - y1);

   while (1) {
      printf("\n1. Translation\n2. Scaling\n3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &c);

      switch (c) {
      case 1:
         printf("Enter the translation factors (tpx, tpy): ");
         scanf("%f%f", &tpx, &tpy);

         x11 = x1 + tpx;
         y11 = y1 + tpy;
         x22 = x2 + tpx;
         y22 = y2 + tpy;
         x33 = x3 + tpx;
         y33 = y3 + tpy;

         setcolor(RED);
         line(midx + x11, midy - y11, midx + x22, midy - y22);
         line(midx + x22, midy - y22, midx + x33, midy - y33);
         line(midx + x33, midy - y33, midx + x11, midy - y11);
         break;

      case 2:
         printf("Enter the scaling factors (sfx, sfy): ");
```

```
          scanf("%f%f", &sfx, &sfy);

           x11 = x1 * sfx;
           y11 = y1 * sfy;
           x22 = x2 * sfx;
           y22 = y2 * sfy;
           x33 = x3 * sfx;
           y33 = y3 * sfy;

           setcolor(YELLOW);
           line(midx + x11, midy - y11, midx + x22, midy - y22);
           line(midx + x22, midy - y22, midx + x33, midy - y33);
           line(midx + x33, midy - y33, midx + x11, midy - y11);
           break;

        case 3:
           closegraph();
           exit(0);

        default:
           printf("Invalid choice. Please try again.\n");
        }
     }

     getch();
  }
```

Output



```
2D Translation, Rotation and Scaling
Enter the points (x1,y1), (x2,y2) and (x3,y3) of triangle
50 20
100 20
75 40

 1.Transaction
 2.Rotation
 3.Scalling
 4.exit
Enter your choice:1
Enter the translation factor
20
30
Enter your choice:
```



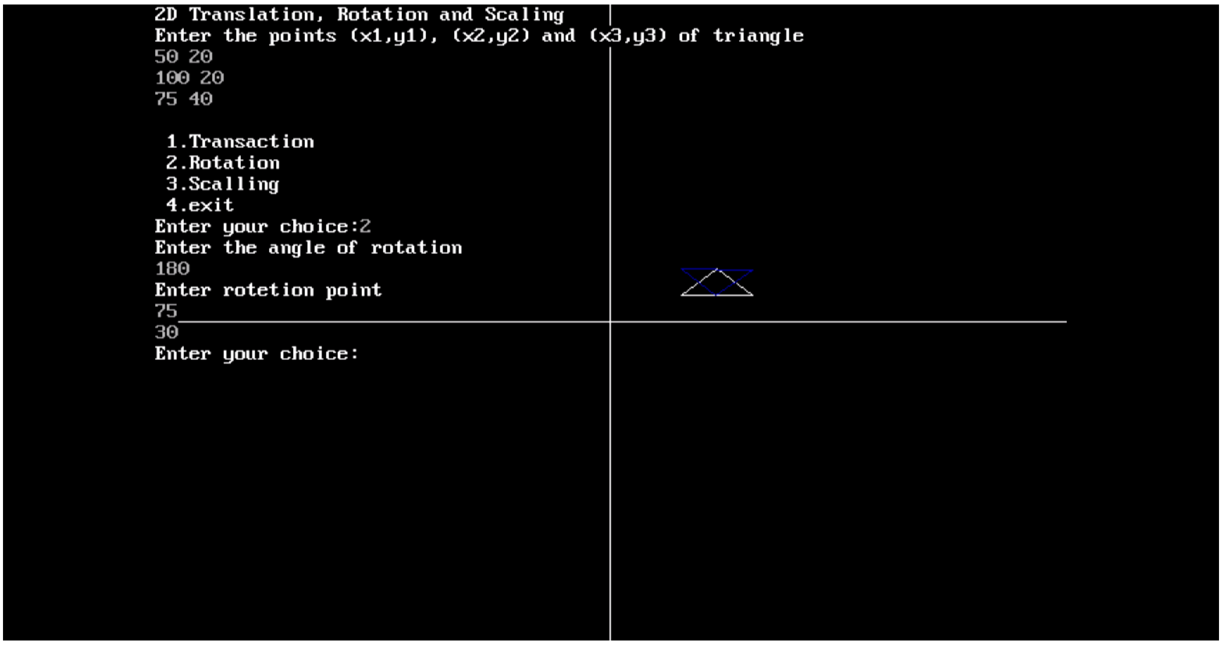
```
2D Translation, Rotation and Scaling
Enter the points (x1,y1), (x2,y2) and (x3,y3) of triangle
50 20
100 20
75 40

 1.Transaction
 2.Rotation
 3.Scalling
 4.exit
Enter your choice:3
Enter the scalling factor
5
5
Enter scaling point
75
30
Enter your choice:
```

7. Write a Program for rotation of the triangle

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

void DrawTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
void RotateTriangle(int x1, int y1, int x2, int y2, int x3, int y3, float angle);

int main()
{
   int gd = DETECT, gm;
   int x1, y1, x2, y2, x3, y3;
   float angle;

   initgraph(&gd, &gm, "");

   printf("Enter the 1st point for the triangle (x1 y1): ");
   scanf("%d%d", &x1, &y1);

   printf("Enter the 2nd point for the triangle (x2 y2): ");
   scanf("%d%d", &x2, &y2);

   printf("Enter the 3rd point for the triangle (x3 y3): ");
   scanf("%d%d", &x3, &y3);

   DrawTriangle(x1, y1, x2, y2, x3, y3);

   printf("Enter the angle for rotation (in degrees): ");
   scanf("%f", &angle);

   RotateTriangle(x1, y1, x2, y2, x3, y3, angle);

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

void DrawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
{
   line(x1, y1, x2, y2);
   line(x2, y2, x3, y3);
   line(x3, y3, x1, y1);
}

void RotateTriangle(int x1, int y1, int x2, int y2, int x3, int y3, float angle)
{
   int p = x2, q = y2;
```

```
    float radianAngle = (angle * 3.14) / 180.0;

    int a1 = p + (x1 - p) * cos(radianAngle) - (y1 - q) * sin(radianAngle);
    int b1 = q + (x1 - p) * sin(radianAngle) + (y1 - q) * cos(radianAngle);

    int a2 = p + (x2 - p) * cos(radianAngle) - (y2 - q) * sin(radianAngle);
    int b2 = q + (x2 - p) * sin(radianAngle) + (y2 - q) * cos(radianAngle);

    int a3 = p + (x3 - p) * cos(radianAngle) - (y3 - q) * sin(radianAngle);
    int b3 = q + (x3 - p) * sin(radianAngle) + (y3 - q) * cos(radianAngle);

    setcolor(1);
    DrawTriangle(a1, b1, a2, b2, a3, b3);
}
```

Output

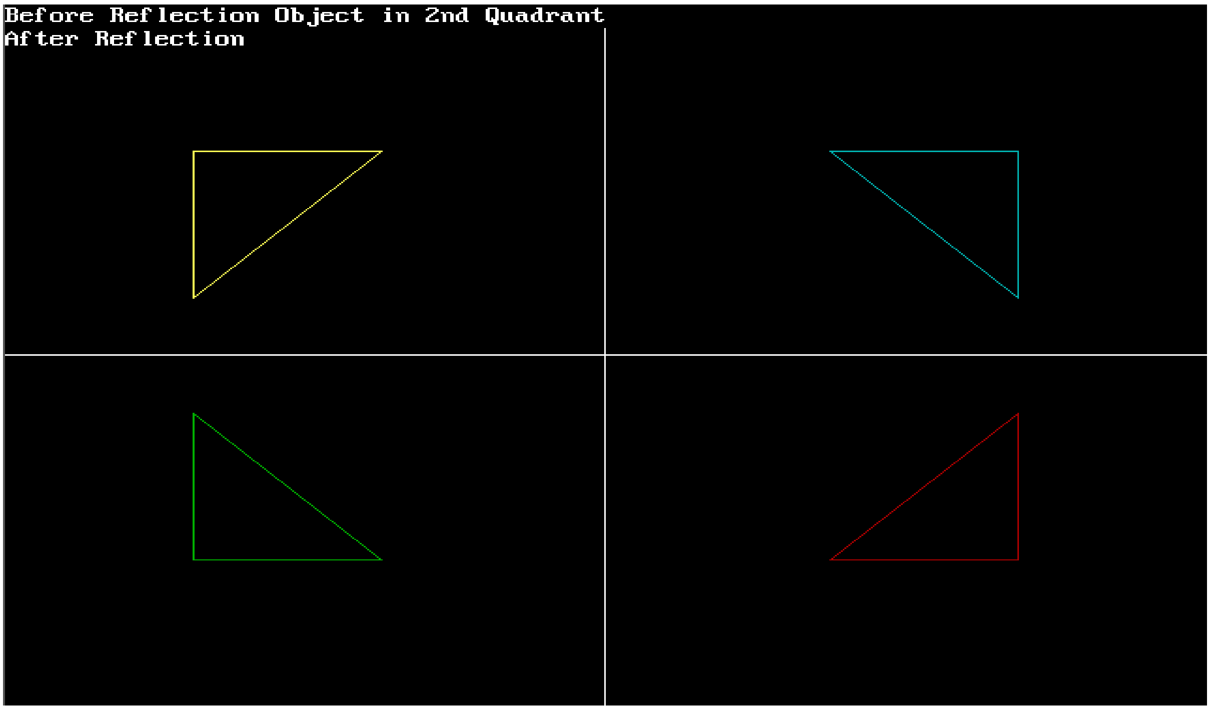


```
2D Translation, Rotation and Scaling
Enter the points (x1,y1), (x2,y2) and (x3,y3) of triangle
50 20
100 20
75 40

 1.Transaction
 2.Rotation
 3.Scalling
 4.exit
Enter your choice:2
Enter the angle of rotation
180
Enter rotetion point
75
30
Enter your choice:
```

8. Write a Program for reflection of the triangle.

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

void main()
{

int gm, gd = DETECT, ax, x1 = 100;
int x2 = 100, x3 = 200, y1 = 100;
int y2 = 200, y3 = 100;

initgraph(&gd, &gm, "");
cleardevice();

line(getmaxx() / 2, 0, getmaxx() / 2,
      getmaxy());
line(0, getmaxy() / 2, getmaxx(),
      getmaxy() / 2);

printf("Before Reflection Object"
      " in 2nd Quadrant");

setcolor(14);
line(x1, y1, x2, y2);
line(x2, y2, x3, y3);
line(x3, y3, x1, y1);
getch();


printf("\nAfter Reflection");

setcolor(4);
line(getmaxx() - x1, getmaxy() - y1,
      getmaxx() - x2, getmaxy() - y2);

line(getmaxx() - x2, getmaxy() - y2,
      getmaxx() - x3, getmaxy() - y3);

line(getmaxx() - x3, getmaxy() - y3,
      getmaxx() - x1, getmaxy() - y1);

setcolor(3);
line(getmaxx() - x1, y1,
      getmaxx() - x2, y2);
line(getmaxx() - x2, y2,
      getmaxx() - x3, y3);
line(getmaxx() - x3, y3,
        getmaxx() - x1, y1);

setcolor(2);
line(x1, getmaxy() - y1, x2,
        getmaxy() - y2);
line(x2, getmaxy() - y2, x3,
        getmaxy() - y3);
line(x3, getmaxy() - y3, x1,
        getmaxy() - y1);
getch();

closegraph();
}
```

Output



```
Before Reflection Object in 2nd Quadrant
After Reflection
```

9. Write a Program for Cohen Sutherland Clipping Algorithm

```
#include <stdio.h>

#define INSIDE 0 // 0000
#define LEFT 1   // 0001
#define RIGHT 2  // 0010
#define BOTTOM 4 // 0100
#define TOP 8    // 1000

const int x_max = 10;
const int y_max = 8;
const int x_min = 4;
const int y_min = 4;
int computeCode(double x, double y)
{
   int code = INSIDE;

   if (x < x_min) // To the left of rectangle
      code |= LEFT;
   else if (x > x_max) // To the right of rectangle
      code |= RIGHT;
   if (y < y_min) // Below the rectangle
      code |= BOTTOM;
   else if (y > y_max) // Above the rectangle
      code |= TOP;

   return code;
}

void cohenSutherlandClip(double x1, double y1, double x2, double y2)
{
   // Compute region codes for P1, P2
   int code1 = computeCode(x1, y1);
   int code2 = computeCode(x2, y2);

   int accept = 0;

   while (1)
   {
      if ((code1 == 0) && (code2 == 0))
      {
         accept = 1;
         break;
      }
      else if (code1 & code2)
      {
         break;
      }
```

```
    else
    {

        int code_out;
        double x, y;

        if (code1 != 0)
            code_out = code1;
        else
            code_out = code2;
        if (code_out & TOP)
        {
            // Point is above the clip rectangle
            x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
            y = y_max;
        }
        else if (code_out & BOTTOM)
        {
            x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
            y = y_min;
        }
        else if (code_out & RIGHT)
        {
            y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
            x = x_max;
        }
        else if (code_out & LEFT)
        {
            y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
            x = x_min;
        }

        if (code_out == code1)
        {
            x1 = x;
            y1 = y;
            code1 = computeCode(x1, y1);
        }
        else
        {
            x2 = x;
            y2 = y;
            code2 = computeCode(x2, y2);
        }
    }
}

if (accept)
{
    printf("Line accepted from %.2f, %.2f to %.2f, %.2f\n", x1, y1, x2, y2);
    }
    else
    {
        printf("Line rejected\n");
    }
}

int main()
{
    cohenSutherlandClip(5, 5, 7, 7);
    cohenSutherlandClip(7, 9, 11, 4);
    cohenSutherlandClip(1, 5, 4, 1);

    return 0;
}
```

Output:

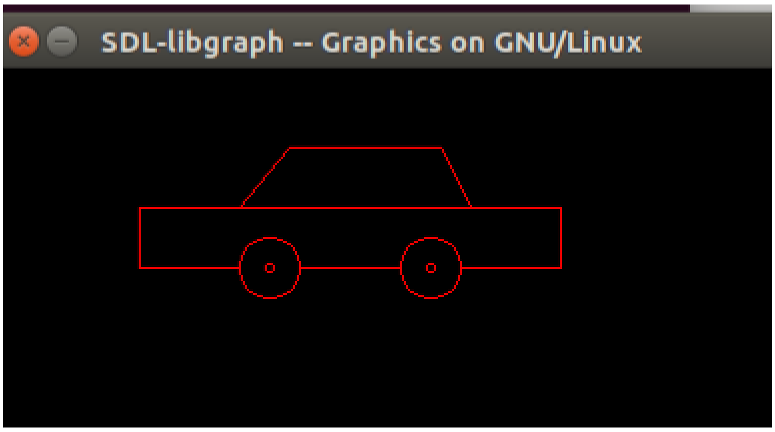
```
Line accepted from 5.00, 5.00 to 7.00, 7.00
Line accepted from 7.80, 8.00 to 10.00, 5.25
Line rejected
```

10.

b.) Develop a program for a moving animated car. You can use the line and circle drawing algorithm for that purpose. The dynamic appearance o the car should be in the animated form.

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

void draw_moving_car(void)
 {

   int i, j = 0, gd = DETECT, gm;

   initgraph(&gd, &gm, "");

   for (i = 0; i <= 420; i = i + 10)
{

      setcolor(RED);

      line(0 + i, 300, 210 + i, 300);
      line(50 + i, 300, 75 + i, 270);
      line(75 + i, 270, 150 + i, 270);
      line(150 + i, 270, 165 + i, 300);
      line(0 + i, 300, 0 + i, 330);
      line(210 + i, 300, 210 + i, 330);

      circle(65 + i, 330, 15);
      circle(65 + i, 330, 2);

      circle(145 + i, 330, 15);
      circle(145 + i, 330, 2);

      line(0 + i, 330, 50 + i, 330);
      line(80 + i, 330, 130 + i, 330);
      line(210 + i, 330, 160 + i, 330);

      delay(100);

      cleardevice();
   }

   getch();

   closegraph();
}

int main()
{
    draw_moving_car();

    return 0;
}
```

Output

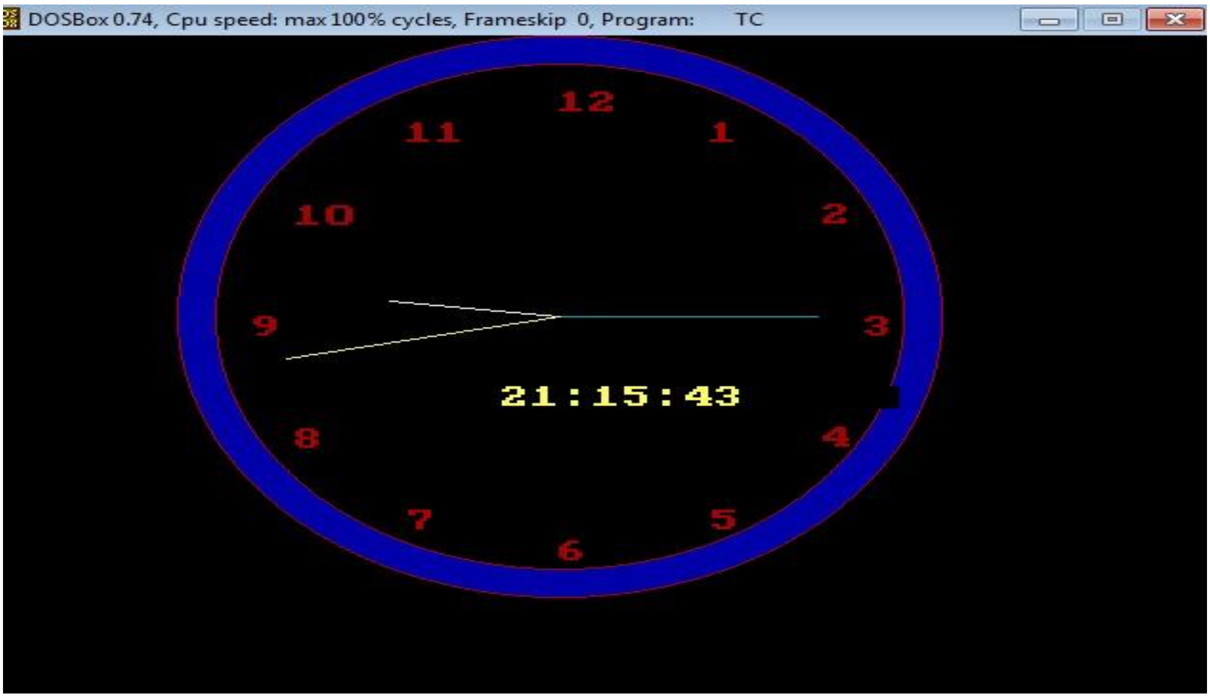


d.) Develop a program to design the analog clock. You can use the line and circle drawing algorithm for that purpose.

```
#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

#include<math.h>

#include<dos.h>

#include<time.h>


#define PI 3.147


void clockLayout();

void secHand();

void hrHand();

void minHand();

int maxx, maxy;


void main()

{

   int gdriver = DETECT, gmode, error;

   initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi\\");

   error = graphresult();

   if (error != grOk)

   {

      printf("Error in graphics, code= %d", grapherrormsg(error));

      exit(0);

   }


   while (1)

   {

      clockLayout();
```

```
        secHand();
        minHand();
        hrHand();
        sleep(1);
        cleardevice();
    }
}

void clockLayout()
{
    int i, x, y, r;
    float j;
    maxx = getmaxx();
    maxy = getmaxy();

    for (i = 1; i < 5; i++)
    {
        setcolor(YELLOW);
        circle(maxx / 2, maxy / 2, 120 - i);
    }

    pieslice(maxx / 2, maxy / 2, 0, 360, 5);
    x = maxx / 2 + 100; y = maxy / 2;
    r = 100;
    setcolor(BLUE);

    for (j = PI / 6; j <= (2 * PI); j += (PI / 6))
    {
        pieslice(x, y, 0, 360, 4);
        x = (maxx / 2) + r * cos(j);
        y = (maxy / 2) + r * sin(j);
    }
    x = maxx / 2 + 100; y = maxy / 2;
    r = 100;
    setcolor(RED);

    for (j = PI / 30; j <= (2 * PI); j += (PI / 30))
    {
        pieslice(x, y, 0, 360, 2);
        x = (maxx / 2) + r * cos(j);
        y = (maxy / 2) + r * sin(j);
    }
}

void secHand()
{
    struct time t;
    int r = 80, x = maxx / 2, y = maxy / 2, sec;
    float O;

    maxx = getmaxx();
    maxy = getmaxy();
    gettime(&t);
    sec = t.ti_sec;
    O = sec * (PI / 30) - (PI / 2);
    setcolor(YELLOW);
    line(maxx / 2, maxy / 2, x + r * cos(O), y + r * sin(O));
}

void hrHand()
{
    int r = 50, hr, min;
    int x, y;
    struct time t;
    float O;

    maxx = getmaxx();
    maxy = getmaxy();
    x = maxx / 2; y = maxy / 2;
    gettime(&t);
    hr = t.ti_hour;
    min = t.ti_min;

    if (hr <= 12) O = (hr * (PI / 6) - (PI / 2)) + ((min / 12) * (PI / 30));
    if (hr > 12) O = ((hr - 12) * (PI / 6) - (PI / 2)) + ((min / 12) * (PI / 30));
    setcolor(BLUE);
    line(maxx / 2, maxy / 2, x + r * cos(O), y + r * sin(O));
}

void minHand()
{
    int r = 60, min;
    int x, y;
    float O;
    struct time t;
    maxx = getmaxx();
    maxy = getmaxy();
    x = maxx / 2;
    y = maxy / 2;
    gettime(&t);
    min = t.ti_min;
    O = (min * (PI / 30) - (PI / 2));
    setcolor(RED);
    line(maxx / 2, maxy / 2, x + r * cos(O), y + r * sin(O));
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
12
1
2
3
4
5
6
7
8
9
10
11
21:15:43

e) Develop a program to clip the image. Show the image before clipping and after the clipping.

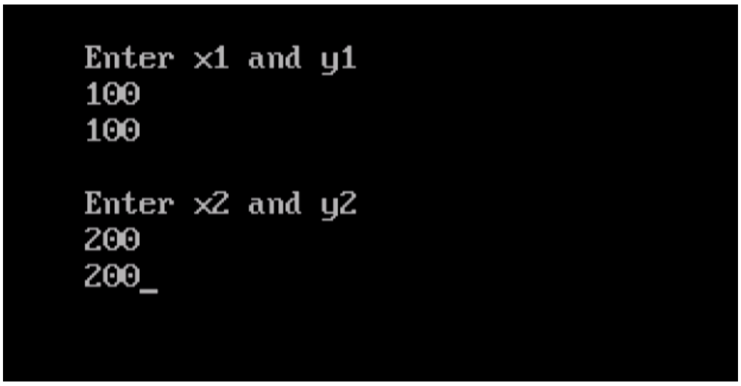
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <graphics.h>
#include <dos.h>
typedef struct coordinate {
    int x, y;
    char code[4];
} PT;


void drawwindow();
void drawline(PT p1, PT p2);
PT setcode(PT p);
int visibility(PT p1, PT p2);
PT resetendpt(PT p1, PT p2);
```

```
void main() {
    int gd = DETECT, gm, v;
    PT p1, p2, p3, p4;

    printf("\nEnter x1 and y1\n");
    scanf("%d %d", &p1.x, &p1.y);
    printf("\nEnter x2 and y2\n");
    scanf("%d %d", &p2.x, &p2.y);

    initgraph(&gd, &gm, "c:\\turboc3\\bgi");
    drawwindow();
    delay(500);

    drawline(p1, p2);
    delay(500);
    cleardevice();

    p1 = setcode(p1);
    p2 = setcode(p2);

    // Check visibility
    v = visibility(p1, p2);
    delay(500);

    switch (v) {
        case 0: // Line is fully visible
            drawwindow();
            delay(500);
            drawline(p1, p2);
            break;
        case 1: // Line is completely invisible
            drawwindow();
            delay(500);
            break;
        case 2: // Line is partially visible (clipping needed)
            p3 = resetendpt(p1, p2);
            p4 = resetendpt(p2, p1);
            drawwindow();
            delay(500);
            drawline(p3, p4);
            break;
    }

    delay(5000);
    closegraph();
}
void drawwindow() {
    line(150, 100, 450, 100);
    line(450, 100, 450, 350);
    line(450, 350, 150, 350);
    line(150, 350, 150, 100);
```

```
}

void drawline(PT p1, PT p2) {
    line(p1.x, p1.y, p2.x, p2.y);
}

PT setcode(PT p) {
    PT ptemp;

    ptemp.code[0] = (p.y < 100) ? '1' : '0'; // Top
    ptemp.code[1] = (p.y > 350) ? '1' : '0'; // Bottom
    ptemp.code[2] = (p.x > 450) ? '1' : '0'; // Right
    ptemp.code[3] = (p.x < 150) ? '1' : '0'; // Left

    ptemp.x = p.x;
    ptemp.y = p.y;

    return ptemp;
}
int visibility(PT p1, PT p2) {
    int i, flag = 0;
    for (i = 0; i < 4; i++) {
        if ((p1.code[i] != '0') || (p2.code[i] != '0'))
            flag = 1;
    }

    if (flag == 0) return 0; // Fully visible

    for (i = 0; i < 4; i++) {
        if ((p1.code[i] == p2.code[i]) && (p1.code[i] == '1'))
            return 1; // Fully invisible
    }

    return 2; // Partially visible
}

PT resetendpt(PT p1, PT p2) {
    PT temp;
    int x, y, i;
    float m, k;

    if (p1.code[3] == '1') // Left
        x = 150;
    if (p1.code[2] == '1') // Right
        x = 450;

    if ((p1.code[3] == '1') || (p1.code[2] == '1')) {
        m = (float)(p2.y - p1.y) / (p2.x - p1.x);
        k = p1.y + m * (x - p1.x);
        temp.y = k;
        temp.x = x;
```

```
            for (i = 0; i < 4; i++)
                temp.code[i] = p1.code[i];

            if (temp.y <= 350 && temp.y >= 100)
                return temp;
        }

        if (p1.code[0] == '1') // Top
            y = 100;
        if (p1.code[1] == '1') // Bottom
            y = 350;

        if ((p1.code[0] == '1') || (p1.code[1] == '1')) {
            m = (float)(p2.y - p1.y) / (p2.x - p1.x);
            k = (float)p1.x + (float)(y - p1.y) / m;
            temp.x = k;
            temp.y = y;

            for (i = 0; i < 4; i++)
                temp.code[i] = p1.code[i];

            return temp;
        } else {
            return p1;
        }
    }
```

Output



```
Enter x1 and y1
100
100

Enter x2 and y2
200
200_
```

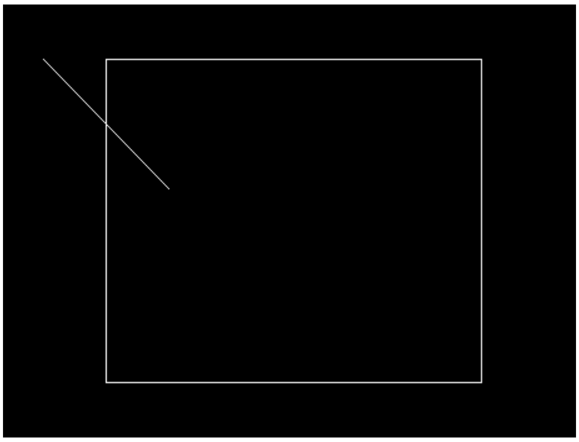


Figure 1Before Clipping

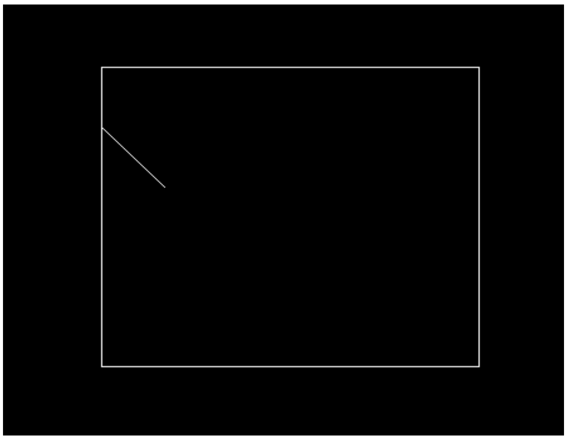


Figure 2After Clipping