INSTITUTE OF TECHNOLOGY
& MANAGEMENT
GWALIOR • MP • INDIA

Laboratory Manual

Computer Graphics & Multimedia
(IT-601)

For
Third Year Students
Department: Information Technology

Department of Information Technology

Vision of IT Department

The Department of Information Technology envisions preparing technically competent problem solvers, researchers, innovators, entrepreneurs, and skilled IT professionals for the development of rural and backward areas of the country for the modern computing challenges.

Mission of the IT Department

- To offer valuable education through an effective pedagogical teaching-learning process.
- To shape technologically strong students for industry, research & higher studies.
- To stimulate the young brain entrenched with ethical values and professional behaviors for the progress of society.

Programme Educational Objectives

Graduates will be able to

- Our engineers will demonstrate application of comprehensive technical knowledge for innovation and entrepreneurship.
- Our graduates will employ capabilities of solving complex engineering problems to succeed in research and/or higher studies.
- Our graduates will exhibit team-work and leadership qualities to meet stakeholder business objectives in their careers.
- Our graduates will evolve in ethical and professional practices and enhance socioeconomic contributions to the society.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes

Computer Graphics & Multimedia (IT 601)

CO1:	Understand the core concepts of computer graphics.
CO2:	Implement various shapes drawing algorithms.
CO3:	Apply geometric transformations on graphic objects and also implement clipping, shading and colour models.
CO4:	Understand multimedia systems architecture, multimedia components and use various multimedia tools.
CO5:	Perform activities involved in design, development and testing of modeling, rendering, shading and animation.

Course	Course Outcomes	CO Attainment	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	Understand the core concepts of computer graphics.		2	0	0	0	0	0	0	0	0	0	0	0	1		1
CO2	Implement various shapes drawing algorithms.		0	2	0	1	1	0	0	0	1	0	0	0	1		1
CO3	Understand multimedia systems architecture, multimedia components and use various multimedia tools.		1	1	2	1	2	0	0	0	0	0	0	0	2	1	0
CO4	Understand multimedia systems architecture, multimedia components and use various multimedia tools.		1	1	0	1	0	0	0	0	0	0	0	0	1		1
CO5	Perform activities involved in design, development and testing of modeling, rendering, shading and animation.		0	1	2	0	1	1	0	0	1	0	0	0	2	1	0

List of Program

S. No.	List	Course Outcome	Page No.
1.	Write a C Program to develop DDA Line Algorithm	CO3	1-2
2.	Write a program to implement Bresenham's line drawing algorithm.	CO3	3-5
3.	Write a program to implement Bresenham's circle drawing algorithm.	CO3	6-8
4.	WAP to implement midpoint circle algorithm.	CO2,CO4	9-11
5.	Write a c program to draw ellipse using Bresenham's Algo.	CO4	12-14
6.	Write a program to perform various transformations on line , square & rectangle.	CO4,CO3	15-19
7.	Write a program to implement Cohen Sutherland line clipping algorithm**.** .	CO4	20-24
8.	Write a program to implement Liang-Bersky line clipping algorithm.	CO3,CO4	25-27
9.	Write a program to implement Cohen-Sutherland polygon clipping algorithm to clip a polygon with a Pattern**.**	CO3,CO5	28-31
10. .	Write a program to convert a color given in RGB space to it's equivalent CMY color space.	CO3,CO5	32-34

1. Write a C Program to develop DDA Line Algorithm.

Student should get the knowledge of creation of line by DDA Line Algo.

DDA Algorithm:

Step1: Start Algorithm

Step2: Declare x_1,y_1,x_2,y_2,dx,dy,x,y as integer variables.

Step3: Enter value of x_1,y_1,x_2,y_2.

Step4: Calculate dx = x_2-x_1

Step5: Calculate dy = y_2-y_1

Step6: If ABS (dx) > ABS (dy)
Then step = abs (dx)
Else

Step7: x_{inc}=dx/step
y_{inc}=dy/step
assign x = x_1
assign y = y_1

Step8: Set pixel (x, y)

Step9: x = x + x_{inc}
y = y + y_{inc}
Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until x = x_2

Step11: End Algorithm

Outcomes: Student will be get the knowledge of creation of line by DDA Line Algorithm.

Program-:

```
1. #include<graphics.h>
2. #include<conio.h>
3. #include<stdio.h>
```

```
4.  void main()
5.  {
6.      intgd = DETECT ,gm, i;
7.      float x, y,dx,dy,steps;
8.      int x0, x1, y0, y1;
9.      initgraph(&gd, &gm, "C:\\TC\\BGI");
10.     setbkcolor(WHITE);
11.     x0 = 100 , y0 = 200, x1 = 500, y1 = 300;
12.     dx = (float)(x1 - x0);
13.     dy = (float)(y1 - y0);
14.     if(dx>=dy)
15.          {
16.        steps = dx;
17.     }
18.     else
19.          {
20.        steps = dy;
21.     }
22.     dx = dx/steps;
23.     dy = dy/steps;
24.     x = x0;
25.     y = y0;
26.     i = 1;
27.     while(i<= steps)
28.     {
29.        putpixel(x, y, RED);
30.        x += dx;
31.        y += dy;
32.        i=i+1;
33.     }
34.     getch();
35.     closegraph();
36. }
```

Output-:

Output:

2. WAP to implement Bresenham's line algorithm.

Objectives: Student should get the knowledge of implement Bresenham's line algorithm.

Outcomes: Student will be get the knowledge of implement Bresenham's line algorithm.

Bresenham's Line Algorithm:

Step1: Start Algorithm

Step2: Declare variable x1,x2,y1,y2,d,i1,i2,dx,dy

Step3: Enter value of x1,y1,x2,y2
Where x1,y1are coordinates of starting point
And x2,y2 are coordinates of Ending point

Step4: Calculate dx = x2-x1
Calculate dy = y2-y1
Calculate i1=2*dy
Calculate i2=2*(dy-dx)
Calculate d=i1-dx

Step5: Consider (x, y) as starting point and xendas maximum possible value of x.
If dx < 0
Then x = x2
y = y2
xend=x1
If dx > 0
Then x = x1
y = y1
xend=x2

Step6: Generate point at (x,y)coordinates.

Step7: Check if whole line is generated.
If x > = xend
Stop.

Step8: Calculate co-ordinates of the next pixel
If d < 0
Then d = d + i1
If d ≥ 0
Then d = d + i2
Increment y = y + 1

Step9: Increment x = x + 1

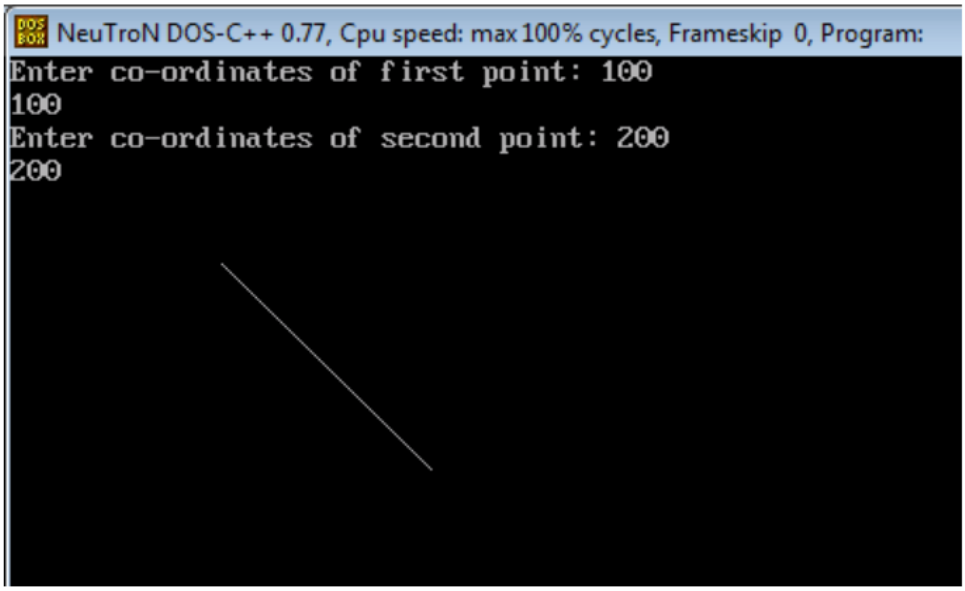
Step10: Draw a point of latest (x, y) coordinates

Step11: Go to step 7

Step12: End of Algorithm

Program

```
#include<stdio.h>
#include<graphics.h>
void drawline(int x0, int y0, int x1, int y1)
{
int dx, dy, p, x, y;
dx=x1-x0;
dy=y1-y0;
x=x0;
y=y0;
p=2*dy-dx;
while(x<x1)
{
if(p>=0)
{
a. putpixel(x,y,7);
b. y=y+1;
c. p=p+2*dy-2*dx;
}
else
{
a. putpixel(x,y,7);
b. p=p+2*dy;}
c. x=x+1;
}
}
int main()
{
int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
printf("Enter co-ordinates of first point: ");
scanf("%d%d", &x0, &y0);
printf("Enter co-ordinates of second point: ");
scanf("%d%d", &x1, &y1);
drawline(x0, y0, x1, y1);
return 0;
}
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program:
Enter co-ordinates of first point: 100
100
Enter co-ordinates of second point: 200
200

Program-3

WAP to implement Bresenham's circle algorithm.

Objectives: Student should get the knowledge implement Bresenham's circle algorithm.

Outcomes: Student will be developed line using Bresenham circle algorithm.

Algorithm:

Bresenham's Circle Algorithm:

1. Step1: Start Algorithm

2. Step2: Declare p, q, x, y, r, d variables
3. p, q are coordinates of the center of the circle
4. r is the radius of the circle

5. Step3: Enter the value of r

6. Step4: Calculate d = 3 - 2r

7. Step5: Initialize x=0
8. &nbsy= r

9. Step6: Check if the whole circle is scan converted
a. If x > = y
b. Stop

10. Step7: Plot eight points by using concepts of eight-way symmetry. The center is at (p, q). Current active pixel is (x, y).
a. putpixel (x+p, y+q)
b. putpixel (y+p, x+q)
c. putpixel (-y+p, x+q)
d. putpixel (-x+p, y+q)
e. putpixel (-x+p, -y+q)
f. putpixel (-y+p, -x+q)
g. putpixel (y+p, -x+q)
h. putpixel (x+p, -y-q)

11. Step8: Find location of next pixels to be scanned
a. If d < 0
b. then d = d + 4x + 6
c. increment x = x + 1
d. If d ≥ 0

e. then d = d + 4 (x - y) + 10
f. increment x = x + 1
g. decrement y = y – 1
12. Step9: Go to step 6
13. Step10: Stop Algorithm

Program:-

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

  void   EightWaySymmetricPlot(int xc,int yc,int x,int y)
 {
  putpixel(x+xc,y+yc,RED);
  putpixel(x+xc,-y+yc,YELLOW);
  putpixel(-x+xc,-y+yc,GREEN);
  putpixel(-x+xc,y+yc,YELLOW);
  putpixel(y+xc,x+yc,12);
  putpixel(y+xc,-x+yc,14);
  putpixel(-y+xc,-x+yc,15);
  putpixel(-y+xc,x+yc,6);
 }

  void BresenhamCircle(int xc,int yc,int r)
 {
  int x=0,y=r,d=3-(2*r);
  EightWaySymmetricPlot(xc,yc,x,y);

  while(x<=y)
   {
    if(d<=0)
          {
     d=d+(4*x)+6;
    }
   else
    {
     d=d+(4*x)-(4*y)+10;
     y=y-1;
    }
     x=x+1;
     EightWaySymmetricPlot(xc,yc,x,y);
    }
```

```
}

int  main(void)
{
/* request auto detection */
int xc,yc,r,gdriver = DETECT, gmode, errorcode;
/* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

 /* 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 */
 }
   printf("Enter the values of xc and yc :");
   scanf("%d%d",&xc,&yc);
   printf("Enter the value of radius  :");
   scanf("%d",&r);
   BresenhamCircle(xc,yc,r);

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

4. WAP to implement midpoint circle algorithm.

Objectives: Student should get the knowledge of implement midpoint circle algorithm.

Outcomes: Student will be developed circle by midpoint circle algorithm.

Algorithm:-

Step1: Put x =0, y =r in equation 2
We have p=1-r

Step2: Repeat steps while x ≤ y
Plot (x, y)
If (p<0)
Then set p = p + 2x + 3
else
p = p + 2(x-y)+5
y =y - 1 (end if)
x =x+1 (end loop)

Step3: End

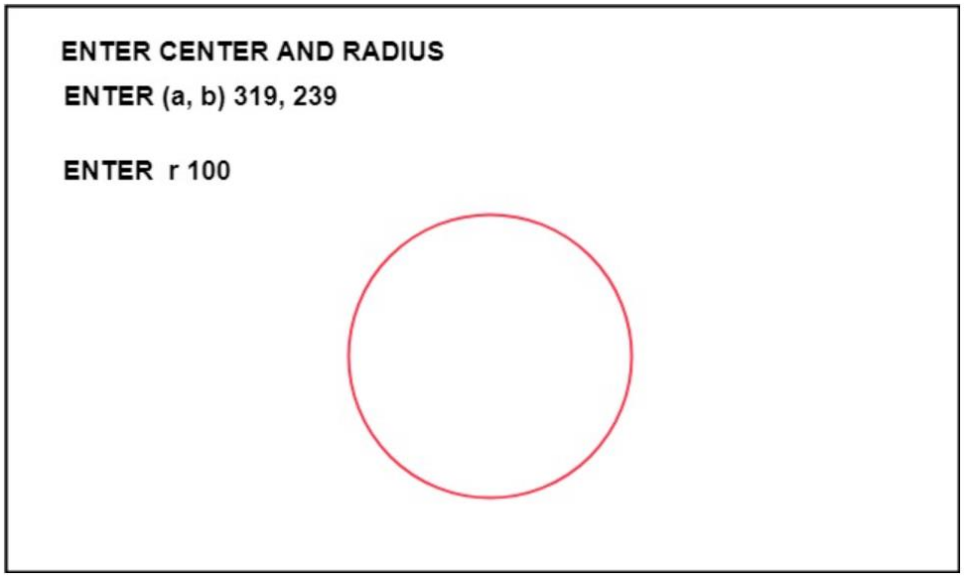
Program:-

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

lass 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);
}
}
```

Output:-



5. Write a c program to draw ellipse using Bresenham's Algo.

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

void drawEllipse(int a, int b) {
    int x = 0, y = b;
    int a_sqr = a * a;
    int b_sqr = b * b;
    int two_a_sqr = 2 * a_sqr;
    int two_b_sqr = 2 * b_sqr;
    int four_a_sqr = 4 * a_sqr;
    int four_b_sqr = 4 * b_sqr;
    int d = b_sqr - a_sqr * b + a_sqr / 4;
    int dx = 0, dy = four_a_sqr * y;

    while (dx < dy) {
        printf("(%d, %d)\n", x, y);
        x++;
        dx += two_b_sqr;
        if (d < 0)
            d += dx + b_sqr;
        else {
            y--;
            dy -= two_a_sqr;
            d += dx - dy + b_sqr;
        }
    }

    d = b_sqr * (x + 0.5) * (x + 0.5) + a_sqr * (y - 1) * (y - 1) - a_sqr * b_sqr;

    while (y >= 0) {
        printf("(%d, %d)\n", x, y);
        y--;
        dy -= two_a_sqr;
        if (d > 0)
            d += a_sqr - dy;
        else {
            x++;
            dx += two_b_sqr;
            d += dx - dy + a_sqr;
        }
    }
}
```

```
int main() {
    int a, b;
    printf("Enter the semi-major axis (a) and semi-minor axis (b) of the ellipse: ");
    scanf("%d %d", &a, &b);
    drawEllipse(a, b);
    return 0;
}
```

Explanation

1. Input:
   - The program takes input from the user for the semi-major axis (a) and semi-minor axis (b) of the ellipse.

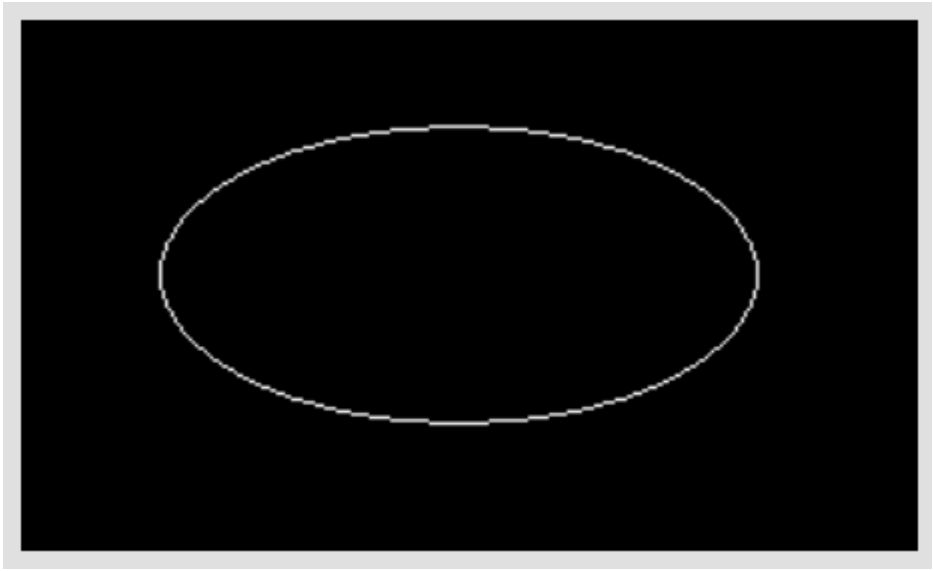
2. Initialization:
   - Initialize variables `x` and `y` to 0 and `b`, respectively, representing the initial point on the ellipse.
   - Compute squared values `a_sqr` (a * a) and `b_sqr` (b * b).
   - Calculate constants `two_a_sqr` (2 * a_sqr), `two_b_sqr` (2 * b_sqr), `four_a_sqr` (4 * a_sqr), and `four_b_sqr` (4 * b_sqr) for optimization.

3. Bresenham's Algorithm:
   - Use Bresenham's algorithm to iterate over points on the ellipse and print them.
   - The algorithm uses a decision parameter `d` to determine the next point based on the current point's position relative to the ellipse's boundary.

4. Drawing the Ellipse:
   - The first while loop handles the region where the slope of the ellipse is less than -1 (dy > dx).
   - The second while loop handles the remaining region where the slope of the ellipse is greater than or equal to -1 (dy <= dx).
   - The algorithm calculates `d` based on the distance of the current point from the ellipse's boundary using the Bresenham's algorithm formulas.

5. Output:
   - The program prints the coordinates of points on the ellipse using `printf("(%d, %d)\n", x, y)`.

6. Main Function:
   - The `main` function takes user input for the semi-major and semi-minor axes.
   - It then calls the `drawEllipse` function with these inputs to draw the ellipse.



6. Write a program to perform various transformations on line , square & rectangle.

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

void drawLine(int x1, int y1, int x2, int y2) {
   line(x1, y1, x2, y2);
}

void drawSquare(int x, int y, int side) {
   rectangle(x, y, x + side, y + side);
}

void drawRectangle(int x, int y, int width, int height) {
   rectangle(x, y, x + width, y + height);
}

void translate(int *x, int *y, int tx, int ty) {
   *x += tx;
   *y += ty;
}

void rotate(int *x, int *y, float angle) {
   float rad = angle * (3.14159 / 180);
   int tempX = *x;
   *x = (int)(*x * cos(rad) - *y * sin(rad));
   *y = (int)(tempX * sin(rad) + *y * cos(rad));
}

void scale(int *x, int *y, float scaleX, float scaleY) {
   *x = (int)(*x * scaleX);
   *y = (int)(*y * scaleY);
}

void reflectX(int *x, int *y) {
   *y = -*y;
}

void reflectY(int *x, int *y) {
   *x = -*x;
}

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

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

// Drawing original shapes
setcolor(WHITE);
drawLine(100, 100, 200, 200);
drawSquare(250, 100, 100);
drawRectangle(400, 100, 150, 80);

// Performing transformations
// Translation
setcolor(RED);
int tx = 50, ty = 50;
translate(&tx, &ty, 100, 100);
drawLine(100 + tx, 100 + ty, 200 + tx, 200 + ty);

setcolor(GREEN);
tx = 50, ty = 50;
translate(&tx, &ty, 250, 100);
drawSquare(250 + tx, 100 + ty, 100);

setcolor(BLUE);
tx = 50, ty = 50;
translate(&tx, &ty, 400, 100);
drawRectangle(400 + tx, 100 + ty, 150, 80);

// Rotation
setcolor(YELLOW);
float angle = 45.0;
int x1 = 100, y1 = 100, x2 = 200, y2 = 200;
rotate(&x1, &y1, angle);
rotate(&x2, &y2, angle);
drawLine(x1, y1, x2, y2);

// Scaling
setcolor(MAGENTA);
float scaleX = 1.5, scaleY = 0.8;
int sq_x = 250, sq_y = 100;
scale(&sq_x, &sq_y, scaleX, scaleY);
drawSquare(sq_x, sq_y, 100);

// Reflection
setcolor(CYAN);
int rx = 100, ry = 100;
reflectX(&rx, &ry);
drawLine(100 + rx, 100 + ry, 200 + rx, 200 + ry);
```

```
    // Delay to view the output
    delay(5000);
    closegraph();
    return 0;
}
```

here's an explanation of the C program that performs various transformations (translation, rotation, scaling, and reflection) on a line, square, and rectangle using the graphics.h library:

1. Include Libraries:
   - The program includes the standard input/output library `stdio.h` for basic I/O operations and the `graphics.h` library for graphics-related functions.

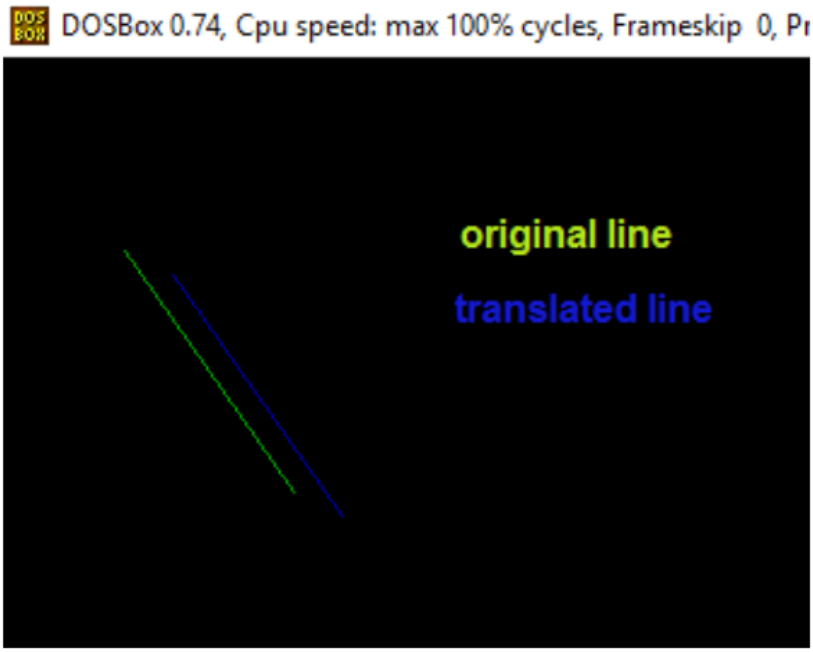
2. Drawing Functions:
   - `drawLine`: Draws a line using the `line` function from `graphics.h`.
   - `drawSquare`: Draws a square using the `rectangle` function from `graphics.h`.
   - `drawRectangle`: Draws a rectangle using the `rectangle` function from `graphics.h`.

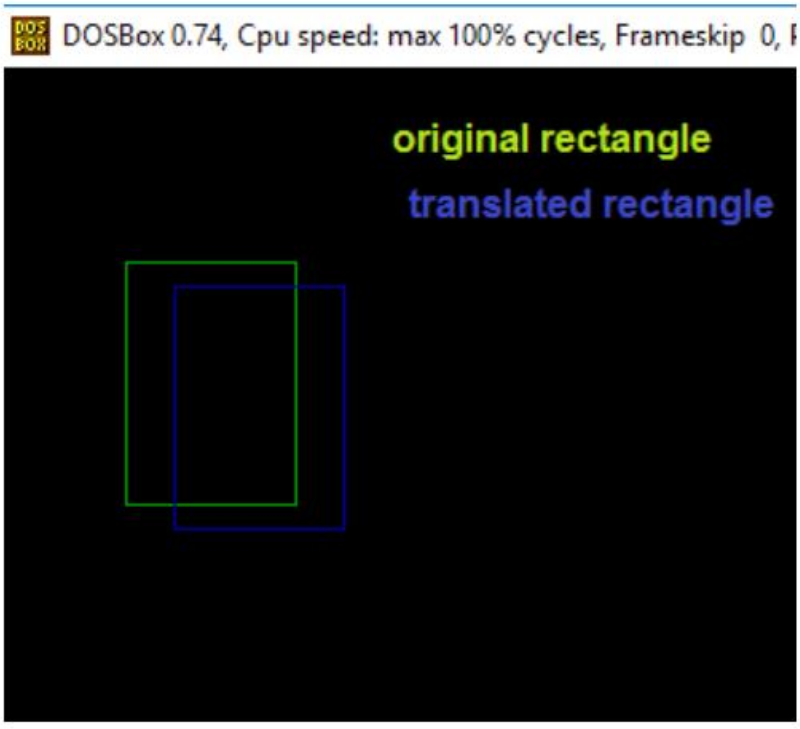
3. Transformation Functions:
   - `translate`: Translates a point by adding the translation values to its coordinates.
   - `rotate`: Rotates a point around the origin by a given angle in degrees using rotation formulas.
   - `scale`: Scales a point by multiplying its coordinates with scaling factors.
   - `reflectX` and `reflectY`: Reflects a point about the X-axis or Y-axis by changing the sign of the corresponding coordinate.

4. Main Function:
   - Initializes the graphics system using `initgraph`.
   - Draws original shapes (a line, square, and rectangle) in white color.
   - Performs various transformations on each shape:
     - Translation of the line, square, and rectangle.
     - Rotation of the line.
     - Scaling of the square.
     - Reflection about the X-axis of the line.
   - Sets different colors for each transformation to distinguish them visually.
   - Uses delay to keep the graphics window open for 5 seconds (`delay(5000)`) and then closes the graphics system (`closegraph()`).

5. Explanation of Transformations:
   - Translation moves shapes by adding or subtracting values from their coordinates.
   - Rotation rotates a shape around the origin based on a specified angle.
   - Scaling changes the size of a shape by multiplying its coordinates with scaling factors.
   - Reflection about the X-axis or Y-axis flips a shape horizontally or vertically.

Overall, the program demonstrates how to use graphics functions in C to draw and transform basic shapes using translation, rotation, scaling, and reflection operations.


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Pr
original line
translated line


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, F
original rectangle
translated rectangle

7. Write a program to implement Cohen Sutherland line clipping algorithm.

Objectives: Student should get the implementation of Cohen Sutherland line clipping algorithm.

Outcomes: Student will be get knowledge of Cohen Sutherland line clipping algorithm.

```
#include <stdio.h>

// Define region codes
const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Define the clipping window
const int x_min = 50;
const int y_min = 50;
const int x_max = 100;
const int y_max = 100;

// Function to compute the region code for a point (x, y)
int computeCode(int x, int y) {
   int code = INSIDE;

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

   return code;
}

// Cohen-Sutherland clipping algorithm
void cohenSutherlandClip(int x1, int y1, int x2, int 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)) {
```

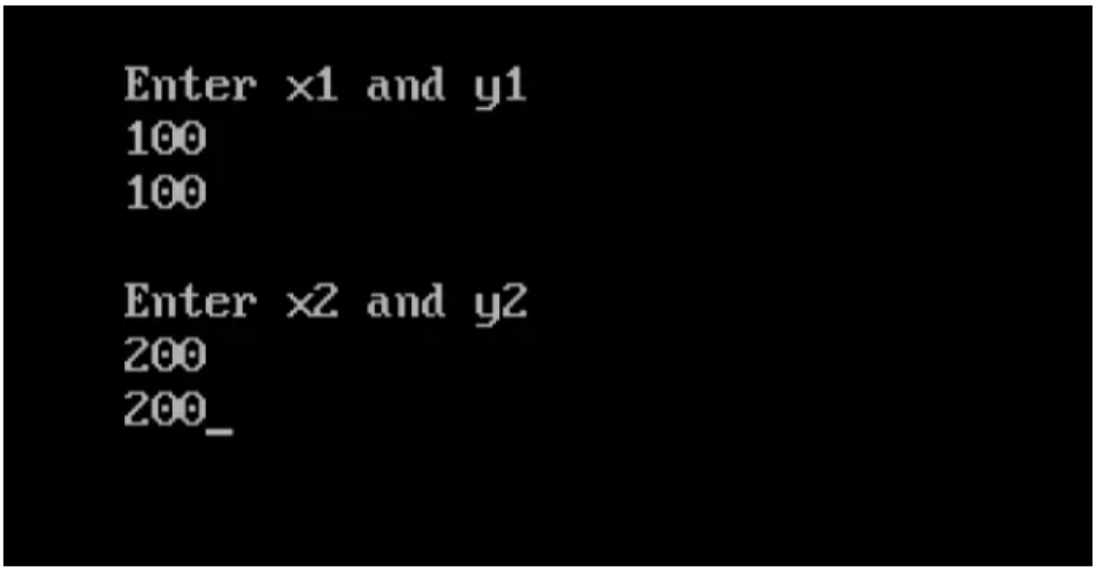
```
            // Both endpoints lie inside the rectangle
            accept = 1;
            break;
        } else if (code1 & code2) {
            // Both endpoints are outside the rectangle in the same region
            break;
        } else {
            // Some segment of the line lies within the rectangle
            int code_out;
            int x, y;

            // At least one endpoint is outside the rectangle, pick it
            if (code1 != 0)
                code_out = code1;
            else
                code_out = code2;

            // Find the intersection point
            if (code_out & TOP) {           // point is above the rectangle
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
                y = y_max;
            } else if (code_out & BOTTOM) { // point is below the rectangle
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
                y = y_min;
            } else if (code_out & RIGHT) {  // point is to the right of rectangle
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
                x = x_max;
            } else if (code_out & LEFT) {   // point is to the left of rectangle
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
                x = x_min;
            }

            // Replace the point outside the rectangle with the intersection point
            // and re-compute the region code
            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 (%d, %d) to (%d, %d)\n", x1, y1, x2, y2);
        // Here you would draw the line from (x1, y1) to (x2, y2)
    } else {
```

```
        printf("Line rejected\n");
    }
}

int main() {
    int x1, y1, x2, y2;
    printf("Enter the coordinates of the line (x1 y1 x2 y2): ");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    cohenSutherlandClip(x1, y1, x2, y2);
    return 0;
}
```

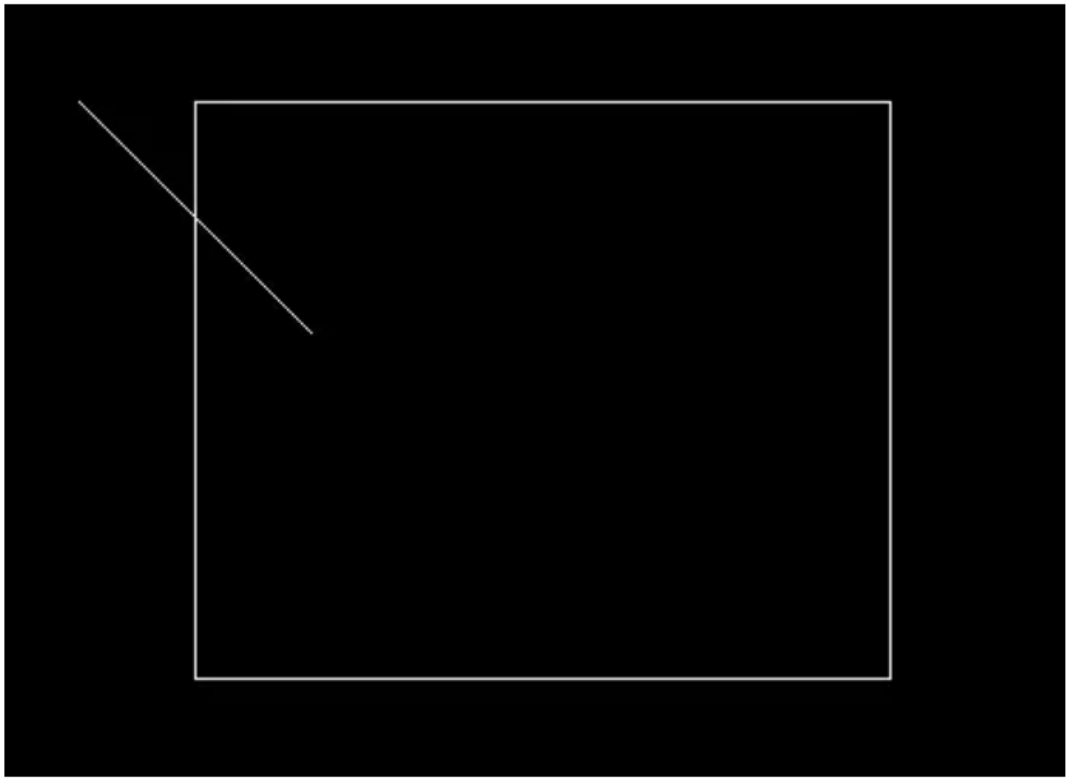
Explanation:

Region Codes: These are used to encode the position of a point relative to the clipping window.
Clipping Window: Defined by x_min, y_min, x_max, and y_max.
computeCode(): This function calculates the region code for a point.
cohenSutherlandClip(): This function implements the Cohen-Sutherland line clipping algorithm. It repeatedly clips the line segment until it either is trivially accepted (both endpoints inside) or rejected (both endpoints share an outside region).

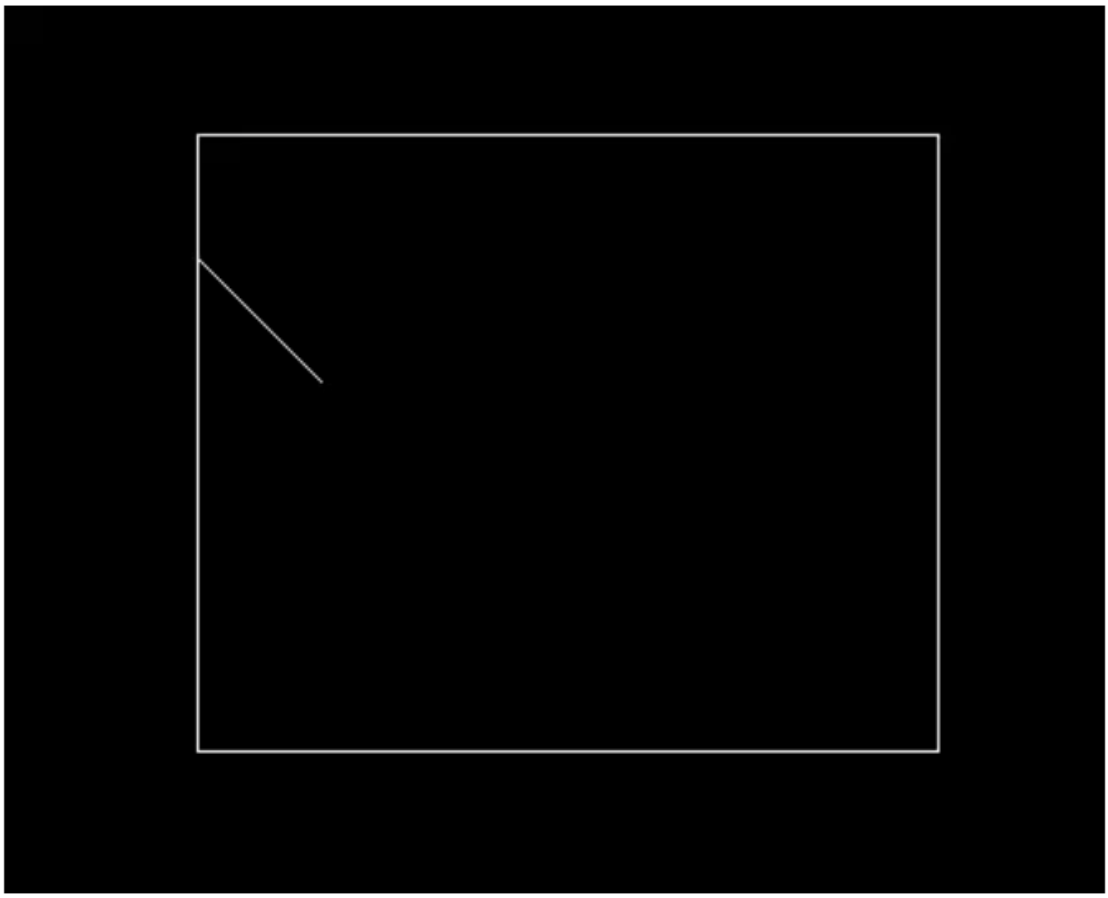
Output



Before Clipping



After Clipping



8. Write a program to implement Liang-Bersky line clipping algorithm.

Objectives: Student should get the knowledge of Liang-Bersky line clipping algorithm.

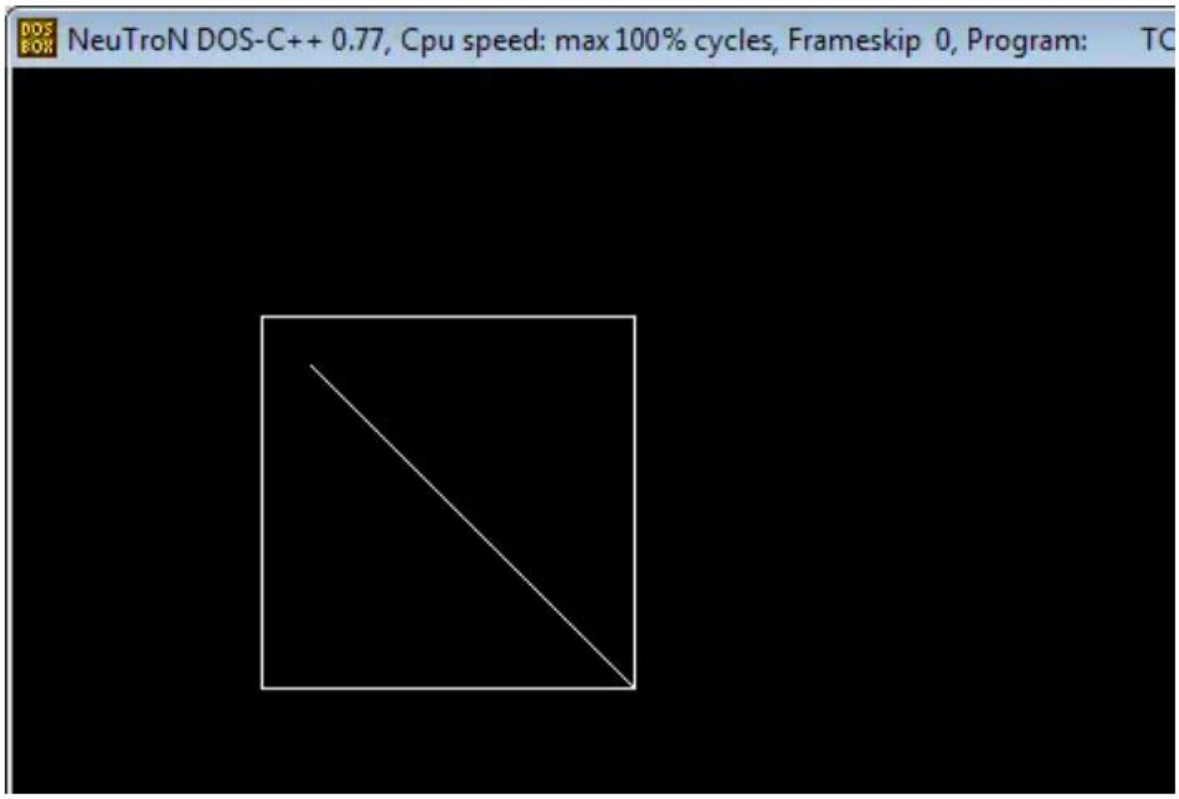
Outcomes: Student will be developed program of Liang-Bersky line clipping algorithm.

```
#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();
}
```

Output



9. Write a program to implement Cohen-Sutherland polygon clipping algorithm to clip a polygon with a Pattern.

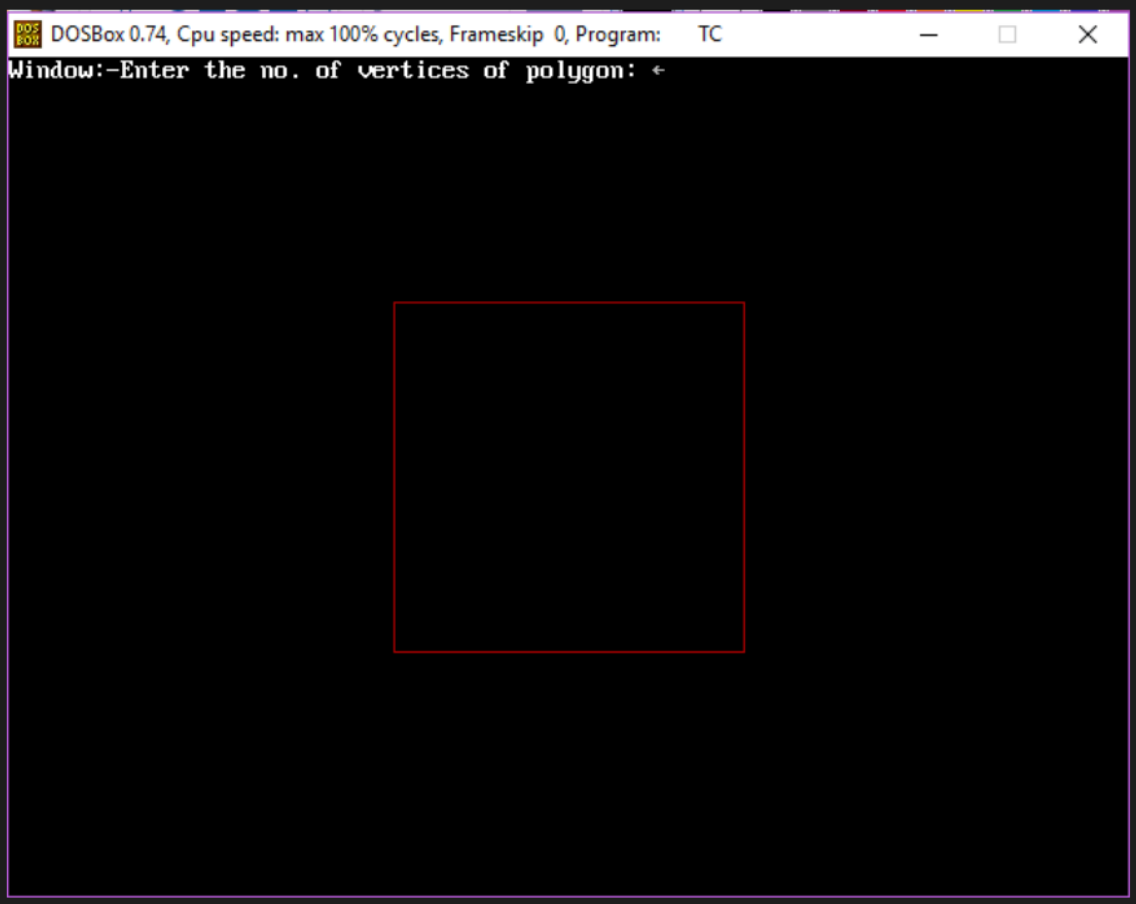
Objectives: Student should get the knowledge of Cohen-Sutherland polygon clipping algorithm.

Outcomes: Student will be developed program of Cohen-Sutherland polygon clipping algorithm.

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
      int gd,gm,n,*x,i,k=0;
      //window coordinates int
      wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;
      int w[]={220,140,420,140,420,340,220,340,220,140};//array for drawing window
      detectgraph(&gd,&gm);
      initgraph(&gd,&gm,"c:\\turboc3\\bgi"); //initializing graphics
      printf("Window:-");
      setcolor(RED); //red colored window
      drawpoly(5,w); //window drawn
      printf("Enter the no. of vertices of polygon: ");
      scanf("%d",&n);
      x = malloc(n*2+1);
      printf("Enter the coordinates of points:\n");
      k=0;
      for(i=0;i<n*2;i+=2) //reading vertices of polygon
      {
      printf("(x%d,y%d): ",k,k);
      scanf("%d,%d",&x[i],&x[i+1]);
      k++;
      }
      x[n*2]=x[0]; //assigning the coordinates of first vertex to last additional vertex for
      drawpoly method.
      x[n*2+1]=x[1];
      setcolor(WHITE);
      drawpoly(n+1,x);
      printf("\nPress a button to clip a polygon..");
      getch();
      setcolor(RED);
      drawpoly(5,w);
```

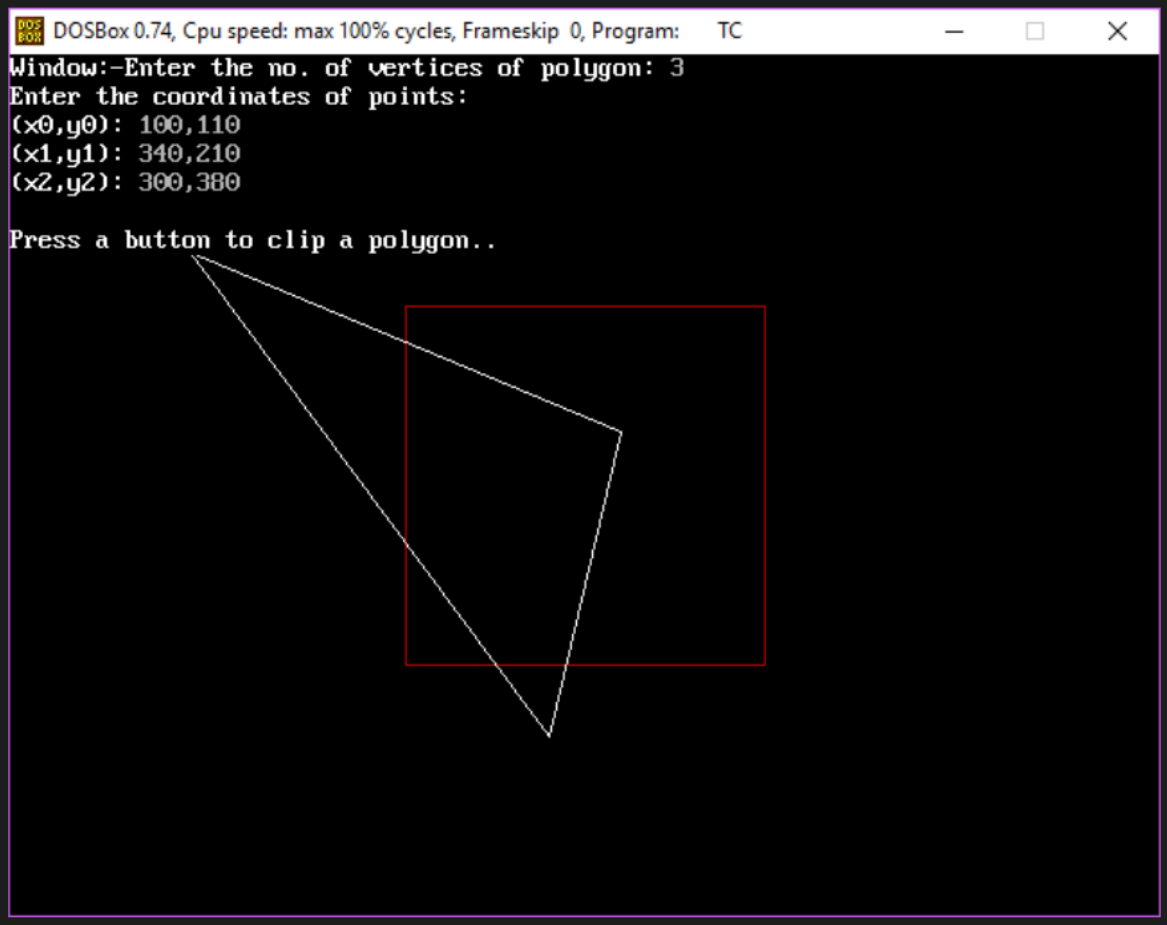
```
        setfillstyle(SOLID_FILL,BLACK);
        floodfill(2,2,RED);
        gotoxy(1,1); //bringing cursor at starting position
        printf("\nThis is the clipped polygon..");
        getch();

        cleardevice();
        closegraph();
        return 0;
}
```

After running this program, first you've to enter the number of vertices of polygon.



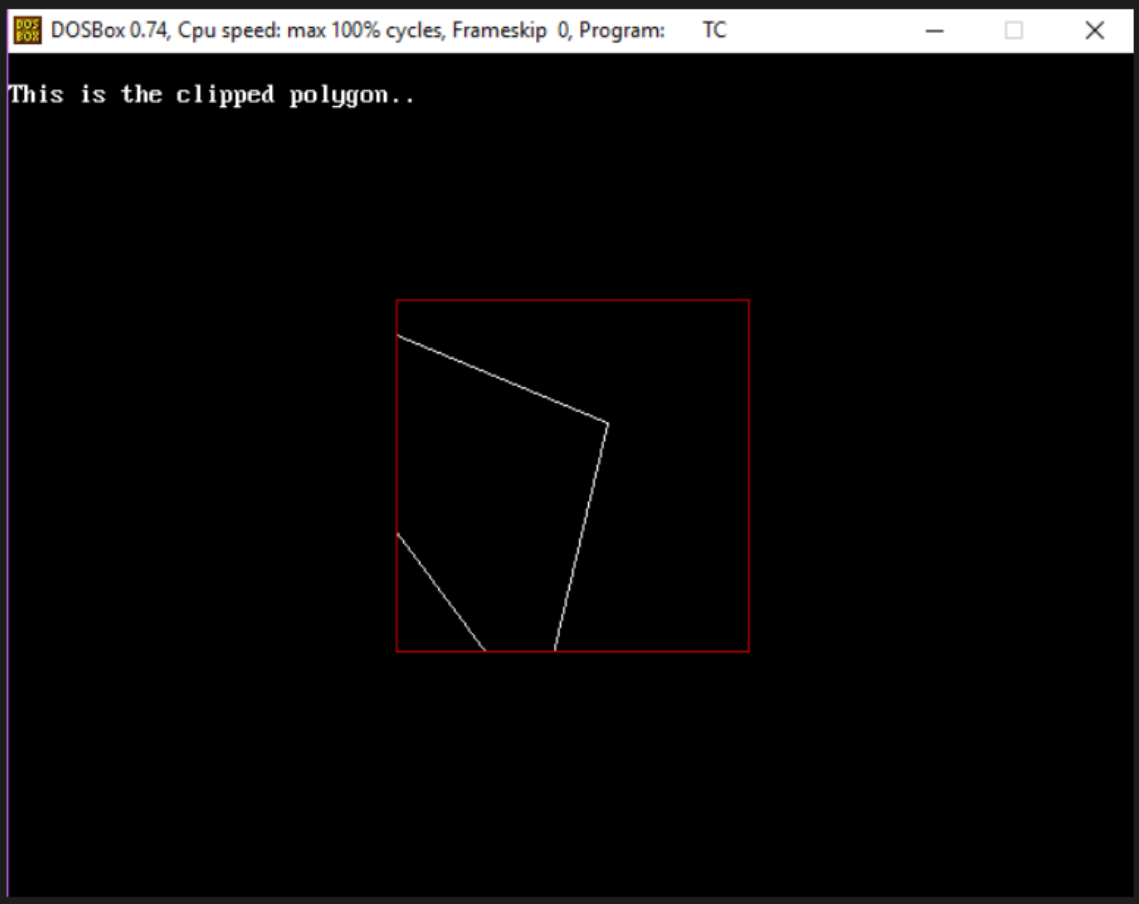
Output: Enter number of vertices

After entering number of vertices you'll be asked to enter coordinates of vertices.



Output: Enter coordinates of vertices

After entering vertices' coordinates just press a button a polygon will be drawn with white color. Now press a button to clip the polygon and you'll simply get the clipped polygon in the output.



Clipped Polygon

So, this is the simplest program to clip a polygon. If you want to change the window coordinates you can change them according to you.

10. Write a program to convert a color given in RGB space to it's equivalent CMY color space.

Objectives: Student should get the knowledge of convert a color given in RGB space to it's equivalent CMY color space.

Outcomes: Student will be developed program of convert a color given in RGB space to it's equivalent CMY color space.

```
#include <stdio.h>

// Function to convert RGB to CMY
void RGBtoCMY(int R, int G, int B, float *C, float *M, float *Y) {
    // Normalize the RGB values to the range [0, 1]
    float r = R / 255.0;
    float g = G / 255.0;
    float b = B / 255.0;

    // Compute the CMY values
    *C = 1 - r;
    *M = 1 - g;
    *Y = 1 - b;
}

int main() {
    int R, G, B;
    float C, M, Y;

    // Get RGB input from user
    printf("Enter the RGB values (0-255):\n");
    printf("R: ");
    scanf("%d", &R);
    printf("G: ");
    scanf("%d", &G);
    printf("B: ");
    scanf("%d", &B);

    // Validate the input
    if ((R < 0 || R > 255) || (G < 0 || G > 255) || (B < 0 || B > 255)) {
        printf("Error: RGB values should be in the range 0-255.\n");
        return 1;
    }

    // Convert RGB to CMY
    RGBtoCMY(R, G, B, &C, &M, &Y);
```

```
    // Print the CMY values
    printf("The CMY values are:\n");
    printf("C: %.2f\n", C);
    printf("M: %.2f\n", M);
    printf("Y: %.2f\n", Y);

    return 0;
}
```

Explanation

1. Function RGBtoCMY: This function takes RGB values and converts them to CMY values.

- It first normalizes the RGB values by dividing by 255.0.
- Then, it calculates the CMY values using the formulas C=1−rC=1−r, M=1−gM=1−g, and Y=1−bY=1−b.

2. Main Function:

- Prompts the user to input RGB values.
- Validates that the input values are within the range 0-255.
- Calls the RGBtoCMY function to perform the conversion.
- Outputs the resulting CMY values..

Input:

```makefile
Enter the RGB values (0-255):
R: 100
G: 150
B: 200
```

Output:

```yaml
The CMY values are:
C: 0.61
M: 0.41
Y: 0.22
```