	(in milliseconds).	delay(10000); /*10000 milliseconds = 10 seconds delay */
getch()	Waits for a keyboard input and returns the ASCII value of the key pressed.	**char ch = getch();**

These functions provide the basic building blocks for creating graphical applications in C. Remember, these functions work with the BGI graphics library, which is typically used in Turbo C++ IDE. If you're using a different compiler or environment, you might need different libraries and functions.

b) Draw a coordinate axis at the centre of the screen.

Solution:

```
#include <graphics.h>
#include <stdlib.h>
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   // Get the center of the screen
   int centerX = getmaxx() / 2;
   int centerY = getmaxy() / 2;
   // Draw X-axis
   line(0, centerY, getmaxx(), centerY);
   // Draw Y-axis
   line(centerX, 0, centerX, getmaxy());
   getch();
   closegraph();
   return 0;
}
```

---

2. Solve the following:

a) Divide your screen into four region, draw circle, rectangle, ellipse and half ellipse in each region with appropriate messages.

Solution:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   int maxX = getmaxx();
```

```
    int maxY = getmaxy();
    // Divide screen into four regions
    int midX = maxX / 2;
    int midY = maxY / 2;
    // Region 1 - Circle
    setcolor(RED);
    circle(midX / 2, midY / 2, 50);
    outtextxy(midX / 2 - 20, midY / 2 + 60, "Circle");
    // Region 2 - Rectangle
    setcolor(GREEN);
    rectangle(midX + 50, midY / 2, midX + 200, midY + 100);
    outtextxy(midX + 70, midY / 2 + 120, "Rectangle");
    // Region 3 - Ellipse
    setcolor(YELLOW);
    ellipse(midX / 2, midY + 50, 0, 360, 100, 50);
    outtextxy(midX / 2 - 20, midY + 110, "Ellipse");
    // Region 4 - Half Ellipse
    setcolor(BLUE);
    pieslice(midX + 150, midY + 150, 0, 180, 100);
    outtextxy(midX + 120, midY + 230, "Half Ellipse");
    getch();
    closegraph();
    return 0;
}
```

b) Draw a simple hut on the screen.

Solution:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   // Draw hut body
  rectangle(150, 200, 400, 400);
  // Draw hut roof
 line(150, 200, 275, 100);
 line(275, 100, 400, 200);
 // Draw door
 rectangle(250, 300, 350, 400);
 // Draw window
 rectangle(180, 250, 270, 330);
 // Draw sun
 circle(50, 50, 30);
```

```
  // Draw grass
  line(0, 400, 640, 400);
  getch();
  closegraph();
  return 0;
}
```

3. **Draw the following basic shapes in the center of the screen :**
   (i) Circle (ii) Rectangle (iii) Square (iv) Concentric Circles (v)Ellipse (vi) Line

Solution :

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
int main() {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");
  int centerX = getmaxx() / 2; // X-coordinate of the center of the screen
  int centerY = getmaxy() / 2; // Y-coordinate of the center of the screen
  // Draw a circle
  int circleRadius = 50;
  circle(centerX, centerY, circleRadius);
  // Draw a rectangle
  int rectWidth = 100;
  int rectHeight = 60;
  rectangle(centerX - rectWidth / 2, centerY - rectHeight / 2, centerX + rectWidth / 2,
centerY + rectHeight / 2);
  // Draw a square
  int squareSize = 80;
  rectangle(centerX - squareSize / 2, centerY - squareSize / 2, centerX + squareSize / 2,
centerY + squareSize / 2);
  // Draw concentric circles
  int numCircles = 5;
  int circleSpacing = 15;
  for (int i = 0; i < numCircles; ++i) {
    circle(centerX, centerY, circleRadius + i * circleSpacing);
  }
  // Draw an ellipse
  int ellipseRadiusX = 70;
  int ellipseRadiusY = 40;
  ellipse(centerX, centerY, 0, 360, ellipseRadiusX, ellipseRadiusY);
  // Draw a line
  int lineLength = 120;
  line(centerX - lineLength / 2, centerY, centerX + lineLength / 2, centerY);
  getch();
  closegraph();
  return 0;
}
```

4 Solve the following:

a) Develop the program for the DDA Line drawing algorithm.

Solution:

```
#include <graphics.h>
#include <math.h>
void drawLineDDA(int x1, int y1, int x2, int y2) {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   int dx = x2 - x1;
   int dy = y2 - y1;
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);
   float xIncrement = (float)dx / steps;
   float yIncrement = (float)dy / steps;
   float x = x1;
   float y = y1;
   for (int i = 0; i <= steps; ++i) {
      putpixel(round(x), round(y), WHITE);
      x += xIncrement;
      y += yIncrement;
   }
   delay(5000);
   closegraph();
}
int main() {
   int x1, y1, x2, y2;
   printf("Enter the starting point (x1 y1): ");
   scanf("%d %d", &x1, &y1);
   printf("Enter the ending point (x2 y2): ");
   scanf("%d %d", &x2, &y2);
   drawLineDDA(x1, y1, x2, y2);
   return 0;
}
```

Note :

Here are some sets of coordinates that you could use to test the DDA line drawing algorithm:

Horizontal Line: (0, 50) to (200, 50)

Vertical Line: (100, 50) to (100, 150)

Diagonal Line - Shallow: (50, 50) to (150, 80)

Diagonal Line - Moderate: (50, 50) to (150, 130)

Diagonal Line - Steep: (50, 50) to (100, 150)

Different Quadrant: (-50, -50) to (100, 150)

b) **Develop the program for Bresenham's Line drawing algorithm.**

Solution:

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

void drawLineBresenham(int x1, int y1, int x2, int y2) {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  int dx = abs(x2 - x1);
  int dy = abs(y2 - y1);

  int slopeGreaterThanOne = 0;

  if (dy > dx) {
    slopeGreaterThanOne = 1;
    int temp = dx;
    dx = dy;
    dy = temp;
  }

  int twoDy = 2 * dy;
  int twoDyMinusDx = 2 * (dy - dx);
  int p = 2 * dy - dx;

  int x, y, xEnd;

  if (x1 > x2) {
    x = x2;
    y = y2;
    xEnd = x1;
  } else {
    x = x1;
    y = y1;
    xEnd = x2;
  }

  if (slopeGreaterThanOne) {
    putpixel(y, x, WHITE);
  } else {
    putpixel(x, y, WHITE);
  }

  while (x < xEnd) {
    x++;

    if (p < 0) {
      p += twoDy;
    } else {
```

```
        y++;
        p += twoDyMinusDx;
      }

      if (slopeGreaterThanOne) {
        putpixel(y, x, WHITE);
      } else {
        putpixel(x, y, WHITE);
      }
    }

    delay(5000);
    closegraph();
}

int main() {
    int x1, y1, x2, y2;
    printf("Enter the starting point (x1 y1): ");
    scanf("%d %d", &x1, &y1);
    printf("Enter the ending point (x2 y2): ");
    scanf("%d %d", &x2, &y2);

    drawLineBresenham(x1, y1, x2, y2);

    return 0;
}
```

Note :

You can try the same input provided in the above program.

---

5. Solve the following:

a) Develop the program for the mid-point circle drawing algorithm.

Solution:

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

void drawCircleMidpoint(int xc, int yc, int radius) {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int x = radius;
    int y = 0;
    int p = 1 - radius;

    while (x > y) {
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
```

```
    putpixel(xc + y, yc - x, WHITE);
    putpixel(xc - y, yc - x, WHITE);
    putpixel(xc + y, yc + x, WHITE);
    putpixel(xc - y, yc + x, WHITE);

    y++;

    if (p <= 0) {
      p = p + 2*y + 1;
    } else {
      x--;
      p = p + 2*y - 2*x + 1;
    }
    if (x < y) {
      break;
    }
  }

  delay(5000);
  closegraph();
}

int main() {
  int xc, yc, radius;
  printf("Enter the center of the circle (x y): ");
  scanf("%d %d", &xc, &yc);
  printf("Enter the radius of the circle: ");
  scanf("%d", &radius);

  drawCircleMidpoint(xc, yc, radius);

  return 0;
}
```

Note:

Here are some sets of input coordinates to test the Midpoint Circle Drawing Algorithm:

Standard Circle: Center at (200, 200) with radius 50
Smaller Circle: Center at (100, 150) with radius 30
Larger Circle: Center at (300, 250) with radius 80
Centered at Origin: Center at (0, 0) with radius 100
Off-Centered Circle: Center at (50, 100) with radius 60

b) Develop the program for the mid-point ellipse drawing algorithm.

Solution:

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

void drawEllipseMidpoint(int xc, int yc, int rx, int ry) {
```

```
int gd = DETECT, gm;
initgraph(&gd, &gm, "");

int x = 0;
int y = ry;
int rxSquare = rx * rx;
int rySquare = ry * ry;
int xChange = 2 * rySquare * x;
int yChange = 2 * rxSquare * y;
int p = rySquare - (rxSquare * ry) + (0.25 * rxSquare);

while (xChange < yChange) {
   putpixel(xc + x, yc + y, WHITE);
   putpixel(xc - x, yc + y, WHITE);
   putpixel(xc + x, yc - y, WHITE);
   putpixel(xc - x, yc - y, WHITE);

   x++;
   xChange += 2 * rySquare;

   if (p < 0) {
      p += rySquare + xChange;
   } else {
      y--;
      yChange -= 2 * rxSquare;
      p += rySquare + xChange - yChange;
   }
}

p = rySquare * (x + 0.5) * (x + 0.5) + rxSquare * (y - 1) * (y - 1) - rxSquare * rySquare;

while (y >= 0) {
   putpixel(xc + x, yc + y, WHITE);
   putpixel(xc - x, yc + y, WHITE);
   putpixel(xc + x, yc - y, WHITE);
   putpixel(xc - x, yc - y, WHITE);

   y--;
   yChange -= 2 * rxSquare;

   if (p > 0) {
      p += rxSquare - yChange;
   } else {
      x++;
      xChange += 2 * rySquare;
      p += rxSquare - yChange + xChange;
   }
}
```

```
delay(5000);
  closegraph();
}

int main() {
  int xc, yc, rx, ry;
  printf("Enter the center of the ellipse (x y): ");
  scanf("%d %d", &xc, &yc);
  printf("Enter the major axis (rx): ");
  scanf("%d", &rx);
  printf("Enter the minor axis (ry): ");
  scanf("%d", &ry);

  drawEllipseMidpoint(xc, yc, rx, ry);

  return 0;
}
```

Note :

Here are some sets of input coordinates to test the Midpoint Ellipse Drawing Algorithm:

Standard Ellipse: Center at (200, 200) with major axis 80 and minor axis 50
Smaller Ellipse: Center at (100, 150) with major axis 40 and minor axis 30
Larger Ellipse: Center at (300, 250) with major axis 120 and minor axis 80
Circular Ellipse: Center at (100, 100) with major axis 60 and minor axis 60
Off-Centered Ellipse: Center at (50, 100) with major axis 100 and minor axis 70

6. Solve the following:

a) Write a program to implement 2D scaling.

Solution:

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

void scale(int x[], int y[], int n, float scaleX, float scaleY) {
  int i;
  for (i = 0; i < n; i++) {
    x[i] = x[i] * scaleX;
    y[i] = y[i] * scaleY;
  }
}

int main() {
  int n, i;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);

  int x[n], y[n];
  printf("Enter the coordinates of the vertices (x y):\n");
```

```
    for (i = 0; i < n; i++) {
        scanf("%d %d", &x[i], &y[i]);
    }

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    // Original polygon
    setcolor(WHITE);
    drawpoly(n, x, y);

    // Scaling factors
    float scaleX, scaleY;
    printf("Enter scaling factor for x: ");
    scanf("%f", &scaleX);
    printf("Enter scaling factor for y: ");
    scanf("%f", &scaleY);

    // Scale the polygon
    scale(x, y, n, scaleX, scaleY);

    // Scaled polygon
    setcolor(RED);
    drawpoly(n, x, y);

    delay(5000);
    closegraph();

    return 0;
}
```

Note:

Here are some suggested inputs to effectively test the scaling functionality of the program:

Square-like Polygon: (0, 0), (100, 0), (100, 100), (0, 100)

Triangle: (50, 50), (100, 100), (150, 50)

Convex Polygon: (50, 50), (100, 100), (150, 200), (200, 150), (150, 50)

For scaling factors:

Equal Scaling Factors: scaleX = 2, scaleY = 2

Different Scaling Factors: scaleX = 1.5, scaleY = 0.5

b) Write a program to perform 2D translation

Solution:

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

void translate(int x[], int y[], int n, int tx, int ty) {
    int i;
    for (i = 0; i < n; i++) {
        x[i] = x[i] + tx;
```

```
    y[i] = y[i] + ty;
  }
}
int main() {
  int n, i;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);

  int x[n], y[n];
  printf("Enter the coordinates of the vertices (x y):\n");
  for (i = 0; i < n; i++) {
    scanf("%d %d", &x[i], &y[i]);
  }

  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  // Original polygon
  setcolor(WHITE);
  drawpoly(n, x, y);

  // Translation factors
  int tx, ty;
  printf("Enter translation in x: ");
  scanf("%d", &tx);
  printf("Enter translation in y: ");
  scanf("%d", &ty);

  // Translate the polygon
  translate(x, y, n, tx, ty);

  // Translated polygon
  setcolor(RED);
  drawpoly(n, x, y);

  delay(5000);
  closegraph();

  return 0;
}
```

Note :

Here are some suggested inputs to test the translation functionality of the program:

Square-like Polygon: (0, 0), (100, 0), (100, 100), (0, 100)

Triangle: (50, 50), (100, 100), (150, 50)

Convex Polygon: (50, 50), (100, 100), (150, 200), (200, 150), (150, 50)

For translation factors:

Positive Translation: tx = 50, ty = 50

Negative Translation: tx = -30, ty = -30

7. Solve the following:

a) Perform 2D Rotation on a given object.

Solution:

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

void rotate(int x[], int y[], int n, float angle) {
   int i;
   float tempX, tempY;
   for (i = 0; i < n; i++) {
      tempX = x[i];
      tempY = y[i];
      x[i] = round(tempX * cos(angle) - tempY * sin(angle));
      y[i] = round(tempX * sin(angle) + tempY * cos(angle));
   }
}

int main() {
   int n, i;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   int x[n], y[n];
   printf("Enter the coordinates of the vertices (x y):\n");
   for (i = 0; i < n; i++) {
      scanf("%d %d", &x[i], &y[i]);
   }

   float angle;
   printf("Enter the rotation angle in degrees: ");
   scanf("%f", &angle);
   angle = angle * (3.1416 / 180.0); // Convert angle from degrees to radians

   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");

   // Original polygon
   setcolor(WHITE);
   drawpoly(n, x, y);

   // Rotate the polygon
   rotate(x, y, n, angle);

   // Rotated polygon
   setcolor(RED);
   drawpoly(n, x, y);

   delay(5000);
```

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

Note :

Here are some suggested inputs to test the rotation functionality of the program:

Square-like Polygon: (0, 0), (100, 0), (100, 100), (0, 100)

Triangle: (50, 50), (100, 100), (150, 50)

Convex Polygon: (50, 50), (100, 100), (150, 200), (200, 150), (150, 50)

For **rotation angles:**

90-Degree Clockwise Rotation: angle = 90

45-Degree Counter-clockwise Rotation: angle = -45

b) **Program to create a house like figure and perform the following operations.**

i) Scaling about the origin followed by translation.

ii) Scaling with reference to an arbitrary point.

iii) Reflect about the line y = mx + c.

Solution:

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

void drawHouse() {
  rectangle(100, 300, 300, 500); // House Base
  line(100, 300, 200, 200);     // Left Roof
  line(200, 200, 300, 300);     // Right Roof
  rectangle(150, 400, 250, 500); // Door
  rectangle(180, 350, 220, 400); // Window
}

void scaleAboutOriginAndTranslate(int sx, int sy, int tx, int ty) {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  setcolor(WHITE);
  drawHouse(); // Original House

  int points[] = {100, 300, 300, 500, 200, 200, 300, 300, 150, 400, 250, 500, 180, 350, 220, 400};
  int n = sizeof(points) / sizeof(points[0]);

  // Scaling about Origin
  for (int i = 0; i < n; i += 2) {
    points[i] *= sx;
    points[i + 1] *= sy;
  }

  // Translation
  for (int i = 0; i < n; i += 2) {
```

```
    points[i] += tx;
    points[i + 1] += ty;
  }

  setcolor(RED);
  drawpoly(n / 2, points);

  delay(2000);
  closegraph();
}

void scaleAboutPoint(int sx, int sy, int px, int py) {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  setcolor(WHITE);
  drawHouse(); // Original House

  // Coordinates of house points
  int points[] = {100, 300, 300, 500, 200, 200, 300, 300, 150, 400, 250, 500, 180, 350, 220, 400};
  int n = sizeof(points) / sizeof(points[0]);

  // Scaling about Arbitrary Point
  for (int i = 0; i < n; i += 2) {
    int x = points[i];
    int y = points[i + 1];
    points[i] = px + (x - px) * sx;
    points[i + 1] = py + (y - py) * sy;
  }

  setcolor(RED);
  drawpoly(n / 2, points);

  delay(2000);
  closegraph();
}

void reflectAboutLine(int m, int c) {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  setcolor(WHITE);
  drawHouse(); // Original House

  // Coordinates of house points
  int points[] = {100, 300, 300, 500, 200, 200, 300, 300, 150, 400, 250, 500, 180, 350, 220, 400};
  int n = sizeof(points) / sizeof(points[0]);

  // Reflect about Line y = mx + c
  for (int i = 0; i < n; i += 2) {
    int x = points[i];
```

```
    int y = points[i + 1];
    int newX = (x + m * (y - c)) / (1 + m * m);
    int newY = 2 * c - y + 2 * m * (x - c) / (1 + m * m);
    points[i] = newX;
    points[i + 1] = newY;
  }

  setcolor(RED);
  drawpoly(n / 2, points);

  delay(2000);
  closegraph();
}

int main() {
  // Scaling about Origin and then Translation
  scaleAboutOriginAndTranslate(2, 2, 100, 100);

  // Scaling about Arbitrary Point
  scaleAboutPoint(2, 2, 150, 400);

  // Reflection about y = mx + c
  reflectAboutLine(1, 100);

  return 0;
}
```

---

8. Solve the following:

a) Write a program to implement Cohen-Sutherland clipping.

Solution:

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

#define LEFT 1   // Bit 0
#define RIGHT 2  // Bit 1
#define BOTTOM 4 // Bit 2
#define TOP 8    // Bit 3

int computeCode(double x, double y, double xmin, double xmax, double ymin, double
ymax) {
  int code = 0;
  if (x < xmin) // to left of window
    code |= LEFT;
  if (x > xmax) // to right of window
    code |= RIGHT;
  if (y < ymin) // below window
    code |= BOTTOM;
  if (y > ymax) // above window
    code |= TOP;
```

```
    return code;
}

void cohenSutherland(double x1, double y1, double x2, double y2, double xmin, double
xmax, double ymin, double ymax) {
    int code1 = computeCode(x1, y1, xmin, xmax, ymin, ymax);
    int code2 = computeCode(x2, y2, xmin, xmax, ymin, ymax);
    int accept = 0;

    while (1) {
        if (!(code1 | code2)) { // Both endpoints inside window
            accept = 1;
            break;
        } else if (code1 & code2) { // Both endpoints outside window, in same region
            break;
        } else {
            int codeOut = code1 ? code1 : code2;
            double x, y;

            if (codeOut & TOP) {
                x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
                y = ymax;
            } else if (codeOut & BOTTOM) {
                x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
                y = ymin;
            } else if (codeOut & RIGHT) {
                y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
                x = xmax;
            } else if (codeOut & LEFT) {
                y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
                x = xmin;
            }

            if (codeOut == code1) {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1, xmin, xmax, ymin, ymax);
            } else {
                x2 = x;
                y2 = y;
                code2 = computeCode(x2, y2, xmin, xmax, ymin, ymax);
            }
        }
    }

    if (accept) {
```

```
    setcolor(GREEN);
    line(x1, y1, x2, y2);
  } else {
    setcolor(RED);
    outtextxy(200, 200, "Line rejected due to clipping!");
  }
}

int main() {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  rectangle(100, 100, 400, 300);  // Clipping window
  cohenSutherland(50, 150, 300, 400, 100, 400, 100, 300);

  delay(5000);
  closegraph();
  return 0;
}
```

b) **Write a program to implement Liang - Barsky Line Clipping Algorithm**

Solution:

```
#include <graphics.h>
#include <stdio.h>
int main() {
  int gd = DETECT, gm;
  float x1, y1, x2, y2, xmin, ymin, xmax, ymax;
  float t1 = 0, t2 = 1, dx, dy;

  printf("Enter the coordinates of the line (x1 y1 x2 y2): ");
  scanf("%f %f %f %f", &x1, &y1, &x2, &y2);
  printf("Enter the coordinates of the rectangle (xmin ymin xmax ymax): ");
  scanf("%f %f %f %f", &xmin, &ymin, &xmax, &ymax);

  dx = x2 - x1;
  dy = y2 - y1;

  int accept = 0;

  if (dx != 0) {
    float t = (xmin - x1) / dx;
    if (dx > 0) {
      if (t > t1) t1 = t;
    } else {
      if (t < t2) t2 = t;
    }

    t = (xmax - x1) / dx;
    if (dx > 0) {
```

```
        if (t < t2) t2 = t;
      } else {
        if (t > t1) t1 = t;
      }
    }

    if (dy != 0) {
      float t = (ymin - y1) / dy;
      if (dy > 0) {
        if (t > t1) t1 = t;
      } else {
        if (t < t2) t2 = t;
      }

      t = (ymax - y1) / dy;
      if (dy > 0) {
        if (t < t2) t2 = t;
      } else {
        if (t > t1) t1 = t;
      }
    }
  if (t1 < t2) {
      int x_start = x1 + t1 * dx;
      int y_start = y1 + t1 * dy;
      int x_end = x1 + t2 * dx;
      int y_end = y1 + t2 * dy;
  initgraph(&gd, &gm, "");
      rectangle(xmin, ymin, xmax, ymax);
      line(x1, y1, x2, y2);
      line(x_start, y_start, x_end, y_end);
  delay(5000);
      closegraph();
    } else {
      printf("Line is completely outside the window.\n");
    }
  return 0;
  }
```

Note :

Here are some suggested inputs to test the Cohen-Sutherland line clipping algorithm:

Line Inside the Window: Line from (50, 50) to (150, 150)

Line Outside the Window: Line from (200, 200) to (300, 300)

Line Crossing the Window: Line from (50, 50) to (200, 200)

Horizontal Line Clipped by Window: Line from (0, 100) to (200, 100)

Vertical Line Clipped by Window: Line from (100, 0) to (100, 200)

9. **Solve the following:**

a) **Write a program to fill a circle using Flood Fill Algorithm.**

Solution:

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

void floodFill(int x, int y, int old_color, int new_color) {
  if (getpixel(x, y) == old_color) {
    delay(1);
    putpixel(x, y, new_color);
    floodFill(x + 1, y, old_color, new_color);
    floodFill(x - 1, y, old_color, new_color);
    floodFill(x, y + 1, old_color, new_color);
    floodFill(x, y - 1, old_color, new_color);
  }
}

int main() {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  // Draw a circle
  circle(200, 200, 50);
  delay(1000);

  // Fill the circle with color YELLOW using Flood Fill Algorithm
  floodFill(200, 200, BLACK, YELLOW);

  delay(5000);
  closegraph();
  return 0;
}
```

b) **Write a program to fill a circle using Boundary Fill Algorithm.**

Solution:

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

void boundaryFill(int x, int y, int fill_color, int boundary_color) {
  if (getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color) {
    delay(1);
    putpixel(x, y, fill_color);
    boundaryFill(x + 1, y, fill_color, boundary_color);
    boundaryFill(x - 1, y, fill_color, boundary_color);
    boundaryFill(x, y + 1, fill_color, boundary_color);
    boundaryFill(x, y - 1, fill_color, boundary_color);
  }
```

```
}
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   // Draw a circle
   circle(200, 200, 50);
   delay(1000);
// Fill the circle with color YELLOW using Boundary Fill Algorithm
   boundaryFill(200, 200, YELLOW, WHITE);
   delay(5000);
   closegraph();
   return 0;
}
```

---

10. Solve the following:

a) Develop a simple text screen saver using graphics functions.

Solution:

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

int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   char text[] = "Welcome to Screensaver!";
   int screenWidth = getmaxx();
   int textWidth = textwidth(text);
   int x = 0, y = getmaxy() / 2;
   while (!kbhit()) {
      cleardevice();
      outtextxy(x, y, text);
      delay(100);
      x += 5;
      if (x > screenWidth - textWidth) {
         x = 0;
      }
   }
   closegraph();
   return 0;
}
```

b) Perform smiling face animation using graphic functions.

Solution:

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

int main() {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  int x = getmaxx() / 2;
  int y = getmaxy() / 2;
  int radius = 50;
  int angle = 0;

  while (!kbhit()) {
    cleardevice();

    // Draw face
    circle(x, y, radius);

    // Draw left eye
    circle(x - 20, y - 20, 5);

    // Draw right eye
    circle(x + 20, y - 20, 5);

    // Draw mouth
    arc(x, y, 225 + angle, 315 - angle, 30);

    delay(100);
    angle += 10;
    if (angle > 30) {
      angle = 0;
    }
  }

  closegraph();
  return 0;
}
```

:) Draw the moving car on the screen.

Solution:

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

nt main() {
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  int x = 50;
  int y = 200;
```

```
while (!kbhit()) {
    cleardevice();

    // Draw car body
    rectangle(x, y, x + 150, y - 50);
    rectangle(x + 10, y - 50, x + 140, y - 90);

    // Draw wheels
    circle(x + 30, y, 15);
    circle(x + 120, y, 15);

    delay(50); // Delay to slow down the animation

    x += 10; // Move the car to the right

    if (x > getmaxx()) {
        x = -150; // Reset car position when it goes off the screen
    }
}

closegraph();
return 0;
}
```