Computer graphics Answers

**1. Write a program to implement Bresenham's line drawing algorithm**.

#include <graphics.h>

#include <conio.h>

#include <iostream>

using namespace std;

void bresenhamLine(int x1, int y1, int x2, int y2) {

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int p = 2 \* dy - dx; // Decision parameter

int twoDy = 2 \* dy;

int twoDyDx = 2 \* (dy - dx);

int x, y, xEnd;

if (x1 > x2) {

x = x2;

y = y2;

xEnd = x1;

} else {

x = x1;

y = y1;

xEnd = x2;

}

putpixel(x, y, WHITE); // Plot the first point

while (x < xEnd) {

x++;

if (p < 0) {

p += twoDy;

} else {

y++;

p += twoDyDx;

}

putpixel(x, y, WHITE); // Plot subsequent points

}}

int main() {

int x1, y1, x2, y2;

cout << "Enter the starting coordinates (x1, y1): ";

cin >> x1 >> y1;

cout << "Enter the ending coordinates (x2, y2): ";

cin >> x2 >> y2;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

bresenhamLine(x1, y1, x2, y2);

getch(); // Wait for key press

closegraph(); // Close the graphics window

return 0;

}

**2. Write a program to implement DDA(Digital Differential Analyzer) line drawing algorithm.**

#include <graphics.h>

#include <conio.h>

#include <iostream>

#include <cmath>

using namespace std;

void DDA\_Line(int x1, int y1, int x2, int y2) {

int dx = x2 - x1;

int dy = y2 - y1;

int steps = max(abs(dx), abs(dy));

float Xinc = dx / (float)steps;

float Yinc = dy / (float)steps;

float x = x1;

float y = y1;

for (int i = 0; i <= steps; i++) {

putpixel(round(x), round(y), WHITE); // Plot the pixel

x += Xinc; // Increment x

y += Yinc; // Increment y

}}

int main() {

int x1, y1, x2, y2;

cout << "Enter the starting coordinates (x1, y1): ";

cin >> x1 >> y1;

cout << "Enter the ending coordinates (x2, y2): ";

cin >> x2 >> y2;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

DDA\_Line(x1, y1, x2, y2);

getch(); // Wait for key press

closegraph(); // Close the graphics window

return 0;}

**3. Polynomial Method of Drawing circle**

#include <graphics.h>

#include <conio.h>

#include <iostream>

#include <cmath>

using namespace std;

void drawCircle(int xc, int yc, int r) {

float x, y;

for (x = -r; x <= r; x += 0.01) {

y = sqrt(r \* r - x \* x);

putpixel(xc + round(x), yc + round(y), WHITE); // Upper half

putpixel(xc + round(x), yc - round(y), WHITE); // Lower half

}}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates (xc, yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

drawCircle(xc, yc, r);

getch(); // Wait for key press

closegraph(); // Close the graphics window

return 0;}

**4. Write a program to implement Midpoint circle drawing algorithm**.

#include <graphics.h>

#include <conio.h>

#include <iostream>

using namespace std;

void plotCirclePoints(int xc, int yc, int x, int 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);

}

void midpointCircleDraw(int xc, int yc, int r) {

int x = 0;

int y = r;

int p = 1 - r; // Initial decision parameter

plotCirclePoints(xc, yc, x, y);

while (x < y) {

x++;

if (p < 0) {

p += 2 \* x + 1;

} else {

y--;

p += 2 \* (x - y) + 1;

}

plotCirclePoints(xc, yc, x, y); // Plot the points

}

}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates (xc, yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

midpointCircleDraw(xc, yc, r);

getch(); // Wait for key press

closegraph(); // Close the graphics window

return 0;

}

**5. Write a program to implement Midpoint line drawing algorithm.**

#include <graphics.h>

#include <conio.h>

#include <iostream>

using namespace std;

void plotLinePoints(int x, int y, int xc, int yc) {

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

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

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

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

}

void midpointLineDraw(int x1, int y1, int x2, int y2) {

int dx = x2 - x1, dy = y2 - y1, dx2 = 2 \* dx, dy2 = 2 \* dy;

int x = x1, y = y1;

bool isSteep = abs(dy) > abs(dx);

if (isSteep) { swap(dx, dy); swap(x, y); }

if (x1 > x2) { swap(x1, x2); swap(y1, y2); }

int p = dy2 - dx;

plotLinePoints(x, y, x1, y1);

for (x = x1; x <= x2; x++) {

if (isSteep) plotLinePoints(y, x, x1, y1);

else plotLinePoints(x, y, x1, y1);

if (p < 0) p += dy2;

else { y += (y2 > y1) ? 1 : -1; p += dy2 - dx2; }

}

}

int main() {

int x1, y1, x2, y2;

cout << "Enter the starting coordinates (x1, y1): ";

cin >> x1 >> y1;

cout << "Enter the ending coordinates (x2, y2): ";

cin >> x2 >> y2;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

midpointLineDraw(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}

**6. Write a program to implement Polygon(Rectangle) filling algorithm.**  
#include <graphics.h>

#include <conio.h>

#include <iostream>

using namespace std;

void fillRectangle(int x1, int y1, int x2, int y2) {

for (int y = y1; y <= y2; y++) {

for (int x = x1; x <= x2; x++) {

putpixel(x, y, WHITE);

}

}

}

int main() {

int x1, y1, x2, y2;

cout << "Enter the top-left corner coordinates (x1, y1): ";

cin >> x1 >> y1;

cout << "Enter the bottom-right corner coordinates (x2, y2): ";

cin >> x2 >> y2;

int gd = DETECT, gm;

initgraph(&gd, &gm, (char\*)"");

fillRectangle(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}

**7. Write a program to implement Midpoint ellipse drawing algorithm.**

#include <graphics.h>

#include <iostream>

#include <math.h>

using namespace std;

void drawEllipse(int rx, int ry, int xc, int yc) {

int x = 0, y = ry;

float rxSq = rx \* rx;

float rySq = ry \* ry;

float dx = 2 \* rySq \* x;

float dy = 2 \* rxSq \* y;

float p1 = rySq - (rxSq \* ry) + (0.25 \* rxSq);

while (dx < dy) {

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

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

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

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

if (p1 < 0) {

x++;

dx = 2 \* rySq \* x;

p1 += dx + rySq;

} else {

x++;

y--;

dx = 2 \* rySq \* x;

dy = 2 \* rxSq \* y;

p1 += dx - dy + rySq;

}

}

float p2 = rySq \* (x + 0.5) \* (x + 0.5) + rxSq \* (y - 1) \* (y - 1) - rxSq \* rySq;

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);

if (p2 > 0) {

y--;

dy = 2 \* rxSq \* y;

p2 -= dy + rxSq;

} else {

y--;

x++;

dx = 2 \* rySq \* x;

dy = 2 \* rxSq \* y;

p2 += dx - dy + rxSq;

}

}

}

int main() {

int rx, ry, xc, yc;

cout << "Enter semi-major axis (rx): ";

cin >> rx;

cout << "Enter semi-minor axis (ry): ";

cin >> ry;

cout << "Enter center x-coordinate: ";

cin >> xc;

cout << "Enter center y-coordinate: ";

cin >> yc;

int gd = DETECT, gm;

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

drawEllipse(rx, ry, xc, yc);

getch();

closegraph();

return 0;

}

**8. Write a program to implement Bresenham's circle drawing algorithm.**#include <graphics.h>

#include <iostream>

using namespace std;

void drawCircle(int xc, int yc, int r) {

int x = 0, y = r;

int d = 3 - 2 \* r;

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);

if (d < 0) {

d = d + 4 \* x + 6;

} else {

d = d + 4 \* (x - y) + 10;

y--;

}

x++;

}

}

int main() {

int xc, yc, r;

cout << "Enter center x-coordinate: ";

cin >> xc;

cout << "Enter center y-coordinate: ";

cin >> yc;

cout << "Enter radius: ";

cin >> r;

int gd = DETECT, gm;

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

drawCircle(xc, yc, r);

getch();

closegraph();

return 0;

}

**9. Write a program to implement 2D transformation(Scaling, Reflection).**

#include <graphics.h>

#include <iostream>

using namespace std;

void drawShape(int points[4][2], int color) {

setcolor(color);

for (int i = 0; i < 3; i++) {

line(points[i][0], points[i][1], points[i + 1][0], points[i + 1][1]);

}

line(points[3][0], points[3][1], points[0][0], points[0][1]);

}

void scaleShape(int source[4][2], int result[4][2], float sx, float sy) {

for (int i = 0; i < 4; i++) {

result[i][0] = source[i][0] \* sx;

result[i][1] = source[i][1] \* sy;

}

}

void reflectShape(int source[4][2], int result[4][2], char axis) {

for (int i = 0; i < 4; i++) {

if (axis == 'x' || axis == 'X') {

result[i][0] = source[i][0];

result[i][1] = -source[i][1];

} else if (axis == 'y' || axis == 'Y') {

result[i][0] = -source[i][0];

result[i][1] = source[i][1];

}

}

}

void shiftShape(int points[4][2], int dx, int dy) {

for (int i = 0; i < 4; i++) {

points[i][0] += dx;

points[i][1] += dy;

}

}

int main() {

int original[4][2] = {

{100, 100},

{200, 100},

{200, 200},

{100, 200}

};

int scaled[4][2], reflectedX[4][2], reflectedY[4][2];

scaleShape(original, scaled, 1.5, 1.5);

reflectShape(original, reflectedX, 'x');

reflectShape(original, reflectedY, 'y');

shiftShape(scaled, 250, 0);

shiftShape(reflectedX, 0, 250);

shiftShape(reflectedY, 250, 250);

int gd = DETECT, gm;

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

drawShape(original, WHITE); // Original: White

drawShape(scaled, GREEN); // Scaled: Green

drawShape(reflectedX, RED); // Reflected on X: Red

drawShape(reflectedY, CYAN); // Reflected on Y: Cyan

getch();

closegraph();

return 0;

}

**10. Write a program to implement 2D transformation (Translation, Rotation).**#include <graphics.h>

#include <iostream>

#include <cmath>

using namespace std;

void drawShape(int points[4][2], int color) {

setcolor(color);

for (int i = 0; i < 3; i++) {

line(points[i][0], points[i][1], points[i + 1][0], points[i + 1][1]);

}

line(points[3][0], points[3][1], points[0][0], points[0][1]);

}

void translateShape(int source[4][2], int result[4][2], int tx, int ty) {

for (int i = 0; i < 4; i++) {

result[i][0] = source[i][0] + tx;

result[i][1] = source[i][1] + ty;

}

}

void rotateShape(int source[4][2], int result[4][2], float angleDegrees, int pivotX, int pivotY) {

float angleRad = angleDegrees \* M\_PI / 180.0;

for (int i = 0; i < 4; i++) {

int x = source[i][0] - pivotX;

int y = source[i][1] - pivotY;

result[i][0] = pivotX + round(x \* cos(angleRad) - y \* sin(angleRad));

result[i][1] = pivotY + round(x \* sin(angleRad) + y \* cos(angleRad));

}

}

int main() {

int original[4][2] = {

{100, 100},

{200, 100},

{200, 200},

{100, 200}

};

int translated[4][2], rotated[4][2];

translateShape(original, translated, 250, 0);

rotateShape(original, rotated, 45, 100, 100); // Rotate 45° around top-left corner

int gd = DETECT, gm;

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

drawShape(original, WHITE); // Original: White

drawShape(translated, GREEN); // Translated: Green

drawShape(rotated, YELLOW); // Rotated: Yellow

getch();

closegraph();

return 0;

}

**11. Write a program to implement Cohen-Sutherland line clipping algorithm.**

#include <graphics.h>

#include <iostream>

using namespace std;

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

int xmin = 100, ymin = 100, xmax = 300, ymax = 200;

int computeCode(int x, int y) {

int code = INSIDE;

if (x < xmin) code |= LEFT;

else if (x > xmax) code |= RIGHT;

if (y < ymin) code |= BOTTOM;

else if (y > ymax) code |= TOP;

return code;

}

// Cohen-Sutherland Line Clipping

void cohenSutherlandClip(int x1, int y1, int x2, int y2) {

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

bool accept = false;

while (true) {

if ((code1 == 0) && (code2 == 0)) {

accept = true;

break;

} else if (code1 & code2) {

break;

} else {

int codeOut;

int x, y;

codeOut = code1 ? code1 : code2;

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);

} else {

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

setcolor(GREEN);

rectangle(xmin, ymin, xmax, ymax);

setcolor(RED);

line(x1, y1, x2, y2); // Clipped line

if (accept) {

setcolor(WHITE);

outtextxy(50, 250, "Line Accepted and Drawn in RED");

} else {

setcolor(WHITE);

outtextxy(50, 250, "Line Rejected");

}

}

int main() {

int x1, y1, x2, y2;

cout<<"Enter the Coordinates : X1,Y1,X2,Y2";

cin>>x1>>y1>>x2>>y2;

setcolor(WHITE);

line(x1, y1, x2, y2); // Original line

int gd = DETECT, gm;

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

cohenSutherlandClip(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}