Experiment :

Date
Page No.

PRACTICAL - 4

Aim : write a program to implement Bresenham's line drawing algorithm.

Theory :

The bresenham's algorithm works by determining the closest pixel to the ideal line at each x - coordinate. The ideal line is defined by the equation $y = mx + b$, where m is the slope of the line and b is the y - intercept.

Decision variable calculation : The decision variable (p) is calculated using the formula :

$$p = 2^* (y_1 - y_0) - (x_1 - x_0)$$

If $P < 0$, the algorithm moves to the next x - coordinate without changing the y - coordinate.

If $P \geq 0$, the algorithm moves to the next x - coordinate and increments the y - coordinate.

Program :

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

Experiment :

Date

Page No.

```
int main(){
    int gdriver = DETECT, gmode;
    int x1, y1, x2, y2, i, dy, dx, p, x, y;

    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    printf("enter the starting coordinates (x1 and y1):");
    scanf("%d %d", &x1, &y1);

    printf(:"Enter the end coordinates (x2 and y2):");
    scanf("%d %d", &x2, &y2);

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

    x = x1;
    y = y1;

    p = 2*dy - dx;
    for(i=0; i<=dx; i++){
        putpixel(x, y, WHITE);
        delay(60);

    if (p<0){
        p = p + 2*dy;
    } else{
        p = p + 2*dy - 2*dx;
        y++;
    }
}
```

Experiment :

Date

Page No.

Practical - 5

Aim : write a program to implement mid point circle drawing algorithm.

Theory :

The mid point circle algorithm is an algorithm used to determine the points needed for rasterizing a circle. we use the mid-point algorithm to calculate all the parameter points of the circle in the first octant and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its center.

Program :

```
#include <stdio.h>
#include <graphics.h>
int main()
{
    int gdriver = DETECT, gmode, xc, yc, r;
    int x, y, d;

    printf("Enter the center coordinates (xc, yc):");
    scanf("%d, %d", &xc, &yc);
    printf("Enter the radius :");
    scanf("%d", &r);
```

Experiment :

Date

Page No.

```
initgraph (&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    x = 0 ;
    y = r ;
    d = 1 - r ;

    while (y > x) {
        if (d > 0) {
            d+ = 2 * x + 3 ;
        } else {
            d+ = 2 * (x - y) + 5 ;
            y -- ;
        }
        x ++ ;

    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 ) ;
        delay (50) ;
    }
    getch() ;
    closegraph () ;
    return 0 ;
}
```

CLASSTIME

Experiment :

Date

Page No.

Practical - 6

Aim : Write a program to implement the cohen - sutherland clipping.

Theory :

cohen - sutherland line clipping is a computer graphics algorithm used to clip line to a rectangular clipping window. It helps determine which portion of a line is visible within a defined viewing area and discard the parts that lie outside.

Program :

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

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

int x_min=100, y_min=100;
int x_max=300, y_max=300;
```

TIME

Experiment :

Date

Page No.

```
int computecode (int x, int y) {
    int code = INSIDE;

    if (x < x-min)
        code |= LEFT;
    else if (x > x-max)
        code |= RIGHT;
    if (y < y-min)
        code |= BOTTOM;
    else if (y > y-max)
        code |= TOP;

    return code;
}

void cohensutherlandclip (int x1, int y1, int x2, int y2)
{
    int code1 = computecode (x1, y1);
    int code2 = computecode (x2, y2);
    int accept = 0;

    while (1) {
        if ((code1 == 0) && (code2 == 0)) {
            accept = 1;
            Break;
```

```
} else {
    int code-out ;
    int x, y ;
if (code 1 | = 0)
    code_out = code 1 ;
else
    code _ out = code 2 ;

if (code_out & top)
{
x = x 1 + (x2 - x1) * (y-max -y 1)/(y2 - y1)
y = y-max ;
} else if (code-out & Bottom){

x = x1 + (x2 - x1) * (y - min - y 1)/(y2 - y1);

set color (RED);
line (x1, y1, x2, y2);

cohen sutherland clip (x1, y1, x2, y2);

get ch ()
closegraph ();
}
```

Experiment :

Date

Page No.

Practical : 7

Aim : write a program to perform 2D rotation on a given object

Program :

```
#include <iostream>
#include <cmath>
using namespace std;

#define PI 3.14159265

struct point {
    float x,y;
};

// function to rotate a point by angle (degrees)

point rotate (point P, float angle){

    float rad = angle * PI / 180.0;
    point newP;
    newP.x = P.x * cos(rad) - P.y * sin(rad);
    newP.y = P.x * sin(rad) + P.y * cos(rad);
    return newP;
}
```

Experiment :

Date

Page No.

```
int main [ ) {
    point A,B,C ;
    float angle ;

// Input triangle points

cout << " Enter coordinates of triangle : \n";
cout << " A(x,y):";
cin >> A.x >> A.y ;
cout << "B (x, y) :";
cin >> B.x >> B.y ;
count << " C (x y) ;";
cin >>(x)>> C.y ;
```

Experiment :

Date

Page No.

Practical : 8

Aim : write a program to perform the smiling face animation using graphic functions

Program :

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

void draw_face(int x, int y, int i, smiling) {
    // face outline
    setcolor(WHITE); // set the wlor for the face border
    setfillstyle(SOLID_FILL, WHITE); // set fill style for the face
    circle(x, y, 50); // face outline.
    floodfill(x, y, WHITE); // fill the face background with white.

// Left eye
setcolor(Black); // set the wlor for the eyes
setfill style(SOLID_FILL, BLACK;
circle(x-20, y-20, 5) // Left eye outline
Flood Fill(x-20, y-20, BLACK);

// Right eye
circle(x+20, y-20, 5);
flood fill(x+20, y-20, BLACK);
```

Experiment :

Date

Page No.

```
void animateface(){
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Neutral face
    cleardevice();
    drawface(200, 200, 0);
    delay(1000);

    // Transition to smiling face
    cleardevice();
    drawface(200, 200, 1);
    delay(1000);

    closegraph();
}
void main(){
    animateface();
}
```

Experiment :

Date

Page No.

Practical - 9

Aim : WAP to draw the moving car on the screen.

Program :

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

// function to draw moving car
void draw_moving_car(void);
    int i, gd = DETECT, gm;

    // initialize graphics mode
    initgraph(&gd, &gm, "C:\\TURBO3\\BGI");

for (i=0; i<=420; i=i+10){
    // set color of car
    setcolor(WHITE);

    // Bonnet and Body of car
    line(0+i, 300, 210+i, 300);
    line(80+i, 300, 75+i, 270);
    line(75+i, 270, 150+i, 270);
    line(150+i, 270, 165+i, 300);
    line(210+i, 300, 210+i, 330);

    // left wheel of car
```

Experiment :

Date

Page No.

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

// Right wheel of car
circle (145+i, 330, 15);
circle (145+i, 330, 2);

// lines connecting wheels
line(0+i, 330, 50+i, 330);
// left to left wheel
line(80+i, 330, 130+i, 330);
// middle of wheels
line(160+i, 330, 210+i; 330); // right of right wheel

delay(100 : // Pause for animation.

// erase previous car.
setcolor(BLACK);

// Bonnet and Body of car (erased)
line(0+i, 300, 210+i, 300);
line(50+i, 300, 75+i; 270);
line(95+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);
```

Experiment :

Date
Page No.

```
// Lines connecting wheels (raised)
line (0+i, 330, 50+i, 330);
line (80+i, 330, 130+i, 330);
line (160+i, 330, 210+i, 330);
}

getch();
closegraph();
}

int main(){
    draw-moving_car();
    return 0;
}
```

Experiment :

Practical - 10

Date

Page No.

Aim : Write a program to display Basic 2D geometric primitives.

Program :

```
# include <graphics.h>
int main(){
    int gd = DELETE, gm;
    int graph(&gd, &gm, NULL);

    setcolor(WHITE);
    outtxy(50, 150, "Line");
    line(80, 200, 250, 200);

    setcolor(WHITE);
    outtextxy(50, 250, "Rectangle");
    rectangle(80, 300, 200, 400);

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