Part II : MCS-053 (Computer Graphics and multimedia)

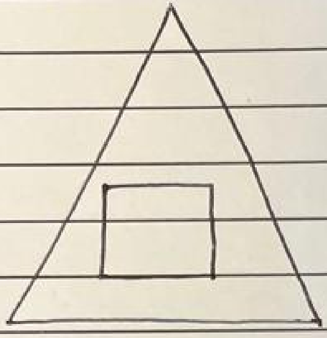
Ques1: Write a program in C/C++ using OpenGL to draw a triangle of orange colour and inside that draw a square of blue colour.

```
# include < Windows. h >
# include <GL/glu.h>
# include <GL/glut.h>
#  include <stdio.h>
#  include <stdlib.h>
# include <math.h>
void drawPoints (void)
 {
double x1 = 100, x2 = 500, x3 = 300, y1 = 100 , y2 =100,
  y3 = 398;
 int midX = (x1+x2+x3)/3;
 int midY = (Y1+ Y2+ Y3)/3;

 /* Clears buffers to preset values */
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3ub (255, 255, 0);
 /* Plot the points */
 gl Begin (GL_LINES);
 gl Line Width (2.5);
 /* Plot the first point */
 gl Vertex 3f (x1, y1, 0);
 gl Vertex 3f (x2, y2, 0);
 gl Begin (GL_LINES);
 gl Line Width (2.5);
 gl Vertex3f (x3, y3, 0);
 gl Vertex 3f (x1, y1, 0);
 gl Begin (GL_LINES);
```

```
glLineWidth (2.5);
glVertex 3f (x2, y2, 0);
gl Vertex3f (x3, y3, 0);

glColor3ub (255, 0,0);
double twice Pi = 2.0* 3.142;
int x = midX, y = midY, i, radius = 95;
gl Begin (GL_LINES); // BEGIN CIRCLE
glVertex2f (midX, midY); // center of circle
for ( i=0; i<= 1000; i++) {
glVertex2f (
 (x+ (radius* cos( i* twice Pi /200))), (y+ (radius*
 sin( i* twicePi /200)))
 );
}

glEnd();
glFlush ();
  }

 void circle()
 {
double x1 = 100, x2 = 500, x3 = 300, y1 = 100, y2 = 100,
      y3 = 398;
double midX = (x1+ x2 + x3)/3;
 double midY = (y1+ y2+ y3)/3;

/* Clears buffers to preset values */
// glClear (GL_COLOR_BUFFER_BIT);
glBegin (GL_POINTS);
// glPointsize (4.5);
gl Vertex2d (midX, midY);
glEnd ();
glFlush ();
```

```
}
void Init()
 {
/* set clear color to white */
glClearColor (0.0, 0.0, 0.0, 0);
/* Set fill color to black */
glColor3f (1.0, 1.0, D.0);
/* glViewport (0,0 , 640, 480); */
/* gl MatrixMode ( GL_PROJECTION); */
/*glLoadIdentity(); */
gluOrtho2D (0, 640, 0, 480);
  }
void main (int argc, char** argv)
{
void main (int argc, char** argv)
 {
glutInit(&argc, argv);
/* set the Initial display mode */
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
/* set the Initial window position and size */
glutInitWindowPosition (0,0);
glutInitWindowSize ( 640, 480);
/* Create the window with title "DDA_Line" */
glutCreateWindow (" Triangle ");
/* Initialize drawing colors */
Init();
/* Call the displaying function */
glutDisplayFunc( drawPoints );
// glutDisplayFunc (circle);
/* Keep displaying until the program is
closed */
 }
```

Output :



Ques 2: Write a program in C/ C++ using OpenGL to draw a hardcoire house as shown in figure given below. Use basic primitives of open GLW.

```
# include < Windows.h >
// for MS Windows
# include <GL\glut.h>
// GLUT, include glu.h and gl.h
// Note: GLglut.h path depending on the system
in use
void init()
 {
// set display window color to as glClearColor
    (R, G, B, Alpha)
glClearColor(0.5, 0.9, 0.4, 0.0);
// set projection parameters.
glMatrixMode ( GL_PROJECTION);
// Set 2D Transformation as gluOrtho2D ( Min
 Width, Max Width, Min Height, Max Height)
gluOrtho2D (0.0, 800, 0.0, 600);
 }
 void Home ()
{
 // Roof
gl Clear (GL_COLOR_BUFFER_BIT); // clear
```

```
 display Window
// set line segment color as glColor 3f (R, G, B)
glColor3f ( 0.3, 0.5, 0.8);
glBegin (GL_POLYGON);
glVertex2i ( 200, 500);
glVertex 2i (600, 500);
glVertex2i ( 700, 350);
glVertex2i (300, 350);
glEnd ();
 // Top of front wall
glColor3f (0.1, 0.5, 0.0);
glBegin (GL- TRIANGLES);
glVertex 2i (200, 500);
glVertex 2i ( 100, 350);
glVertex 2i ( 300, 350);
glEnd ();
 // Front Wall
gl Color 3f (0.7, 0.2, 0.3);
gl Begin (GL_ POLYGON);
glVertex 2i (100, 350);
gl Vertex 2i ( 300, 350);
glVertex2i ( 300, 100);
gl Vertex 2i ( 100, 100) ;
glEnd ();
 // Front Door
gl Color 3f (0.7, 0.2, 0.9);
glBegin (GL_ POLYGON);
gl Vertex 2i ( 150, 250) ;
glVertex 2i ( 250, 250);
gl Vertex 2i (250, 100) ;
glVertex 2i (150, 100) ;
 glEnd ();
```

```
// side Wall
glColor3f(0.1, 0.2, 0.3);
glBegin(GL_POLYGON);
glVertex2i(300, 350);
glVertex2i(700, 350);
glVertex2i(700, 100);
glVertex2i(300, 100);
glEnd();
// window one
glColor3f(0.2, 0.4, 0.3);
glBegin(GL_POLYGON);
glVertex2i(330, 320);
glVertex2i(450, 320);
glVertex2i(450, 230);
glVertex2i(330, 230);
glEnd();
// line of window one
glColor3f(0.1, 0.7, 0.5);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2i(390, 320);
glVertex2i(390, 230);
glVertex2i(330, 273);
glVertex2i(450, 273);
glEnd();
// window two
glColor3f(0.2, 0.4, 0.3);
glBegin(GL_POLYGON);
glVertex x2i(530, 320);
glVertex x2i(650, 320);
glVertex x2i(650, 230);
glVertex x2i(530, 230);
```

```
glEnd();
// Entrance Path
glColor3f(0.3, 0.5, 0.7);
glLineWidth(3);
glBegin(GL_POLYGON);
glVertex2i(150,100);
glVertex2i(250,100);
glVertex2i(210,0);
glVertex2i(40,0);
glEnd();
// Process all OpenGL routine as quickly as
 possible
 glFlush();
  }
int main(int argc, char** argv)
 {
// Initialize GLUT
glutInit(&argc, argv);
// set display mode
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
// set top-left display window position
glutInitWindowPosition(100,100);
// set display window width and height
glutInitWindowSize(800, 600);
// Create display window with the given
title
glutCreateWindow("2D house in OpenGL");
// Execute initialization procedure
init();
// send graphics to display window
glutDisplayFunc(home);
// Display everything and wait.
```

```
glutMainLoop();
 }
```

Ques 3: Write a program in C or C++ to implement Scan-Line Polygon Filling Algorithm.

```
// CPP program to illustrate
// Scanline Polygon fill Algorithm
# include <stdio.h>
# include <math.h>
# include <GL/glut.h>
# define maxHt 800
# define maxWd 600
# define maxVer 10000

FILE *fp;
// Start from lower left corner
typedef struct edgebucket
 {
 int ymax; // max y- coordinate of edge
 float xofymin; // x- coordinate of lowest
 edge point updated only in aet
 float slopeinverse;
 } EdgeBucket;

typedef struct edgetabletup
 {
 // the array will give the scanline number
 // The edge table (ET) with edges entries
 sorted
 // in increasing y and x of the lower end
 int countEdgeBucket; // no. of edgebuckets
 EdgeBucket buckets [maxVer];
} EdgeTableTuple;
EdgeTableTuple EdgeTable [maxHt], ActiveEdgeTuple;

// scanline function
void initEdgeTable()
{
    int i;
   for (i=0; i<maxHt; i++)
      {
         EdgeTable (i). countEdgeBucket = 0;
      }
   ActiveEdgeTuple. countEdgeBucket = 0;
 }

void printTuple (EdgeTableTuple * tup)
 {
     int j;
 if (tup -> countEdgeBucket)
     printf ("\n count %d ---- \n", tup -> buckets [j].
xofymin, tup -> buckets [j]. slopeinverse);
      }
    }

void print table()
  {
     int i, j;
    for ( i=0; i< max Ht; i++)
      {
        if (EdgeTable [i]. countEdgeBucket)
            printf ("\nScanline %d", i);
      printTuple (& EdgeTable [i]);
          }
       }
/* Function to sort an array using insertion
```

```
sort* /
void selectionSort( EdgeTableTuple *ett )
 {
   int i,j;
  EdgeBucket temp;
for ( i=1; i < ett -> buckets[i]. ymax;
 temp. xofymin = ett -> buckets [i]. xofymin;
 temp. slopeinverse = ett -> buckets [i]. slopeinverse;
     j = i-1;
 while ((temp. xofymin < ett -> buckets [j] . xofymin)
   && (j >= 0))
   {
  ett -> buckets [j+1]. ymax = ett -> buckets [j]. ymax;
  ett -> buckets [j+1]. xofymin = ett -> buckets [j].
    xofymin;
ett -> buckets [j+1]. slopeinverse = ett -> buckets [j]
    . slopeinverse;
         j = j -1;
    }
   ett -> buckets [j+1]. ymax = temp. ymax;
   ett -> buckets [j+1]. xofymin = temp. xofymin;
   ett -> buckets [j+1]. slopeinverse = temp. slope
    inverse;
               }
           }
   void storeEdgeInTuple ( EdgeTableTuple *receiver,
     int ym, int xm, float slopInv)
      {
     // both used for edgetable and active
  edge table..
 // the edge tuple sorted in Tsing ymax and
 x of the lower end.
```

```
(receiver-> buckets [(receiver)-> countEdgeBucket]).
     ymax.= ym;
(receiver -> buckets [(receiver)-> countEdgeBucket]).
  xofymin = (float) xm;
(receiver -> buckets [(receiver)-> countEdgeBucket)].
  slopeinverse = slopInv;

  // sort the buckets
insertionSort(receiver);
(receiver -> countEdgeBucket)++;
  }
  void storeEdgeInTable (int x1, int y1, int x2, int y2)
 {
   float m, minv;
   int ymaxTS, xwithyminTS, scanline; //ts
   stands for to store
   if (x2 == x1)
  {
         minv = 0.0000000;
  else
  {
 m = ((float)(y2-y1))/((float)(x2-x1));

 // horizontal lines are not stored in edge table
   if (y2 == y1)

       return;
   minv = (float)1.0/m;
   printf("\nslope string for %d %d & %d
    %d : %f", x1, y1, x2, y2, minv);
  }
  if (y1>y2)
  {
    scanline = y2
      ymaxTS = y1;
    xwithyminTS = x2;
     }
   else
   {
   scanline = y1;
   ymaxTS = y2;
  xwithyminTS = x1;
  }
 // the assignment part is done... now storage...
storeEdgeInTuple (& EdgeTable (scanline], ymaxTS,
xwithyminTS, minV);
     }
 void removeEdgebyYmax (EdgeTableTuple *
     Tup, int yy)
     {
     int i, j;
  for (i=0; i<Tup-> countEdgeBucket; i++)
    {
    if (Tup-> buckets[i]. ymax == yy)
      {
   printf ("\nRemoved at %d", yy);
  for (j=i; j<Tup-> countEdgeBucket-1; j++)
    {
Tup-> buckets[j].ymax = Tup-> buckets[j+1].ymax;
Tup-> buckets[j].xofymin = Tup-> buckets[j+1].
              xofymin;
 Tup-> buckets[j].slopeinverse = Tup-> buckets
            [j+1].slopeinverse;
      }
```

```
            Tup -> countEdgeBucket --;
                    i--;
              }
           }
        }
      void updatexbyslopeinv( Edge Table Tuple * Tup)
        {
            int i;
              for ( i=0; i< Tup -> countEdgeBucket; i++)
           {
    (Tup-> buckets[i]). xofymin = (Tup-> buckets [i]).
           xofymin + (Tup-> buckets [i]). slopeinverse;
         }
           }
        void scanlineFill ()
           {
      /* Follow the following rules:
     1. Horizontal edges: Do not include in edge table.
     2. Horizontal edges: Drawn either on the bottom
          or on the top.
     3. Vertices: If local max or min, then count
              twice, else count once.
     4. Either vertices at local minima or at local
           maxima are drawn. */

    int i, j, x1, ymax1, x2, ymax2, FillFlag =0,
               coordCount;
     // we will start from scanline 0;
        // Repeat until last scanline:
      for ( i=0; i< maxHt; i++) // 4. increment
         y by 1 (next scan line)
      {
```

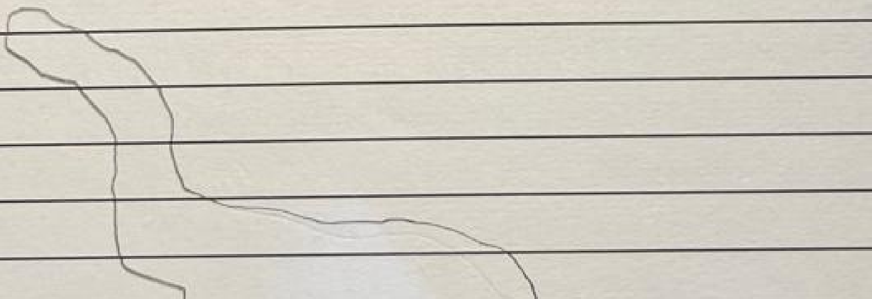
```
//1. Move from ET bucket y to the
// AET those edges whose ymin = y (entering edges)
for ( j=0; j < EdgeTable [i]. count Edge Bucket; j++ )
 {

 storeEdgeInTuple (& ActiveEdgeTuple, EdgeTable [i].
          buckets [j].
   ymax, EdgeTable [i]. buckets [j]. xofymin,
  EdgeTable [i]. buckets [j]. slopeinverse );
 }
 printTuple (& ActiveEdgeTuple);

//2. Remove from AET those edges for // which
    y = ymax (not involved in next scan line)
remove EdgesbyMax (& ActiveEdgeTuple, i);

// sort AET ( remember: ET is presorted )
 insertionsort (& Active EdgeTuple);
 printTuple (& ActiveEdgeTuple );

//3. Fill lines on scan line y by using pairs of
  x- coords from AET
   j = 0;
   Fill flag = 0;
   coord Count = 0;
   x1 = 0;
   x2 = 0;
   ymax1 = 0;
   ymax2 = 0;
 while ( j < Activeedge Tuple. count EdgeBucket)
  {
       if  coordCount % 2 == 0)
   {
    x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);
   ymax1 = ActiveEdgeTuple.buckets[j].ymax;
    if (x1==x2)
  {
   /* three cases can arrive -
 1. lines are towards top of the intersection
 2. lines are towards bottom
 3. one line is towards top and other is
   towards bottom.
  */
 if (((x1 == ymax1)&&(x2!=ymax2))||(x1!=ymax1)
    && (x2 == ymax2)))
   {
        x2 = x1;
        ymax2 = ymax1;
     }
     else
     {
      coordCount++;
      }
   }
     else
     {
         coordCount++;
     }
  }
 else
 {
  x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;
  ymax2 = ActiveEdgeTuple.buckets[j].ymax;
   Fillflag = 0;
  // checking for intersection.....
  if  (x1==x2)
  {
 /* three cases can arive -
1.  lines are towards top of the intersection
2.  lines are towards bottom
3.  one line is towards top and other is
towards bottom.
  */
 if (((x1 == ymax1) && (x2 != ymax2))) || ((x1 != ymax
   1) && (x2 == ymax2))))
   {
      x1 = x2;
      ymax1 = ymax2;
   }
  else
  {
     coordCount++;
      FillFlag = 1;
    }
     }
    else
   {
       coordCount++;
            FillFlag = 1;
     }
  if (FillFlag)
     {
   // drawing actual lines.....
 glColor3f (0.0f, 0.7f, 0.0f);
  glBegin (GL_LINES);
  glVertex2i(x1, i);
```

```
    glVertex2i(x2, i);
    glEnd();
    glFlush();
    // printf("/nLine drawn from %d, %d to %d,
   %d, x1, i, x2, i);
        }
         }
        j++;
    }
 //5. For each nonvertical edge remaining in AET
, update x for new y
 Updatexbyslopeinv(& ActiveEdgeTuple);
   }
 printf("\nScanline filling complete");
  }
  voidmyInit(void)
  {
glClearColor(1.0, 1.0, 1.0, 0.0);
gl MatrixMode( GL_PROJECTION);
gl LoadIdentity();
gluOrtho2D (0, maxHt, 0, maxWd);
glClear( GL_COLOR_BUFFER_BIT);
  }
void drawPolyDino()
  {
glColor3f(1.0f, 0.0f, 0.0f);
intCount = 0, x1, y1, y2, x2;
rewind(fp);
while(!feof(fp))
 {
   count++;
if (count > 2)
```

```
        {
        x1 = x2;
        y1 = y2;
        count = 2;
      }
   if (count ==1)
  {
  fscanf(fp, "%d, %d", &x1, &y1);
    }
   else
   {
   fsanf(fp, "%d, %d", &x2, &y2);
   printf ("\n %d, %d", x2, y2);
  glBegin (GL_LINES);
    glVertex2i(x1, y1);
   glVertex2i(x2, y2);
  glEnd();
storeEdgeInTable (x1, y1, x2, y2); // storage of
edges in edge table.

   glFlush();
     }
   }
    }
  void drawDino( void)
  {
   initEdgeTable();
   drawPolyDino();
 printf ("\nTable");
printTable();
scanlineFill(); // actual calling of scanline filling.
     }
```

```
void main (int argc, char** argv)
  fp = fopen ( "polyDino.txt", "r");
if (fp == Null)
 {
   printf (" could not open file ");
  return;
 }
glutint (& argc, argv);
glutInitDisplayMode ( GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (maxHt, maxWd);
glutInit Window position (100,150);
glutCreateWindow ("Scanline filled dinosaur");
 myInit ();
glutDisplayFunc (drawDino);
 glut MainLoop ();
 fclose(fp);
 }
```

Output :



Ques4: Write a program in C/ C++ to implement Cohen-sutherland line clipping algorithm. In this implementation consider two cases of a line : totally visible, totally invisible, against the rectangular clipping window.

```
//C++ program to implement Cohen Sutherland
  algorithm
// for line clipping
 # include <iostream>
using namespace std;

// Defining 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

// Defining x_max, y_max and x min , y_min for
 // clipping rectangular. since diagonal points
  are
// enough to define a rectangle
const int x_max = 10;
const int y_max = 8;
const int x_min = 4;
const int y_min = 4;

// Function to compute region code for a point
    (x, y)
int computerCode( doublex, double y)
  {
        // initialized as being inside
            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;
          }

    // Implementing Cohen-Sutherland algorithm
     // Clipping a line from P1 = (x2, y2) to P2 =
                                               (x2, y2)
    void cohenSutherlandClip ( double x1, double y1),
                                  double x2, double y2 )
    {
     // computer region codes for P1, P2
     int  code1 =  computecode (x1, y1);
     int  code2 =  computecode (x2, y2);

     // Initialize line as outside the regular window
       bool accept = false;

       while (true)
       {
        if ((code1==0) && (code2==0))
           {
```

```
// if both endpoints lie within rectangle
        accept = true;
   break;
 }
else if ( code1 & code 2)
 {
 // if both endpoints are outside rectangle,
  // in same region
  break;
          }
 else
   {
    // some segment of line lies within the
   // rectangle
   int code - out;
  double x, y;

// At least one endpoint is outside the
 // rectangle , pick it.
if (code1 != 0)
        code_out = code1;
  else
   code_ out = code 2;

// Find intersection point;
// using formulas y = y1 + slope * (x - x1),
// x - x1 + (1 / slope) * (y - y1)
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)
    {
    // 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;
      }
// Now intersection point x, y is found
// We replace point outside rectangle
// by intersection point
if (code_out == code1)

   {
     x1 = x;
     y1 = y;
    code1 = computecode(x1, y1);
      }
    else
        {
          x2 = x;
```

```
            y2 = y;
          code2 = computecode (x2, y2);
            }
          }
      }
      if (accept)
      {
    cout << " line acepted from " << x1 << ", "
        << y1 << " to << x2 << ", "<<y2 << endl;
  // Here the user can add code to display
    the rectangle
  // along with the accepted (portion of) lines
    }
  else
    cout << " line rejected " << endl;
      }
      // Driver code
      int main ()
      {
      // Firstline segment
    // P11 = (5, 5), P12 = (7, 7)
    cohenSutherlandclips (5, 5, 7, 7);

    // second line segment
    // P21 = (7, 9), P22 = (11, 4)
    cohenSutherlandclip ( 7, 9, 11, 4);

  // thirdline segment
  // P31 = (1, 5), P32 = (4, 1)
    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.
```