Graphics Programming in Turbo C/C++

Turbo C++ is one of the very common environments that you might be habituated in for your programming in C/C++. For extraction of graphics utilities of the same, there are a few extra things that you should take care of. For example, there is a header file associated with every graphics manipulation, which is known as "graphics.h". In all graphics programming the user must include this header file.

Before proceeding further, let us first take up the structure of a sample generic graphics program in C to see how elementary things are done here.

```
#include <graphics.h>
#include <stdio.h>
……
…..
main( )
{
int gd=DETECT,gm;
initgraph(&gd,&gm,'\\tcc\\bgi');
…..
…..
closegraph( );
restorecrtmode( );
}
```

In the above program fragment, the function *initgrap ( )* switches over to the graphics mode. The first argument indicates the address of the graphics driver, which directly communicates with the hardware. The number corresponding to the selected graphics mode is put in the next argument. The gm number tells about the monitor, resolution, colors etc. There are different drivers in Turbo C compiler for different monitor systems. However it is left to *initgraph( )* to select the appropriate BGI file (for a particular monitor type). This is done by initializing gd with DETECT which is a macro by itself.

You can really proceed on your own from this stage, if you are aware about some graphics related built-in functions in turbo C/C++ for common use. Accordingly, we now present some of the inbuilt functions supported by Turbo C++ for graphics.

NAME OF THE FUNCTION	BRIEF DESCRIPTION	EXAMPLE

putpixel(x,y)	*Plots the pixel at the grid location (x,y).*	***putpixel(320,240)*** *- plots a pixel with the set color at the center of the screen.*
getpixel(x,y)	*Gets the color of the pixel located at (x,y).*	***getpixel(320,240)*** *- gets the color of the center of the screen for 640X480 resolution.*
line(x1, y1, x2, y2)	*Draws a line from (x1, y1) to (x2, y2).*	***line(0,0,320,240)*** *– draws a line from the upper left of screen to the center.*
circle(xc, yc, r)	*Draws a circle with center (xc, yc) & radius = r.*	***circle(320,240,10)*** *– draws a circle with center at the center of the screen and radius = 10 units.*
rectangle(x1,y1,x2,y2)	*Draws a rectangle with (x1, y1) and (x2, y2) as corners.*	***rectangle(320,240,400, 300)*** *- will draw a rectangle such that (320,240) is the left-top co-ordinate and (400,300) is the right-bottom one.*
ellipse(xc, yc, s, e, xr, yr)	*Draws an ellipse with (xc,yc) as center. xr & yr are semimajor and semiminor axes respectively. If s=0 & e=180 only upper half of the ellipse is drawn.*	***ellipse(320,240,0,360,100,50)*** *– draws an entire ellipse whose center is at (320,240) with semimajor and semiminor axes 100 & 50 respectively.*
arc(xc, yc, s, e, r)	*Draws an arc with center as (xc,yc), radius=r & staring & end angles as s & e respectively.*	***arc(320,240,45,135,100)*** *– draws an arc whose center is at (320,240), radius is 100 units and the start angle is 45 degrees & the end angle 135 degrees.*
moveto(x,y)	*Moves the cursor pointer to (x, y).*	***moveto(320,240) -*** *moves the cursor to the center of the screen from its current location.*
lineto(x,y)	*Draws a line up to (x, y) from the current cursor location.*	***lineto(320,240)*** *– will draw a line from the current location to the center of the screen.*
moverel(xr, yr)	*Moves the cursor by a relative distance of xr along x-axis and yr along y-axis.*	***moverel(10, 5)*** *– moves the cursor in the x-direction by 10 units and in the y-direction by 15 units.*
linerel(xd,yd)	*Draws a line from the cursor point to a point at a relative distance of xd along x and yd along y, from the current position.*	***linerel(10, 5)*** *– draws a line from the current position to a point whose distance from the current position is 10 units along x and 5 units along y.*
outtextxy(x,y,``string'')	*Displays the string within inverted commas at location (x,y).*	***outtextxy(320,240,``Graphics")*** *- displays the word Graphics with the first letter starting from the center of the screen.*

setlinestyle(x,y,z)	*Draws lines with specified styles, where x is the line type, y is the pattern and z is the thickness. (x=0 ⇒ solid, x=1⇒dotted, x=2⇒ center line,x=3⇒ dashed, x=4 ⇒user-defined line).*	***setlinestyle(0,1,1)*** *– sets the system for drawing a solid line with thickness 1 and a particular pattern.*
setcolor(x)	*Sets color of line. There are 16 possible colors.*	***setcolor(RED)*** *– sets the color of the drawing pen as RED.*
setfillstyle(x,y)	*Sets a predefined fill style where x⇒item number identifying a fill style & y⇒color.*	***setfillstyle(SOLID_FILL, RED)*** *– prepares to fill an area in solid style and with RED color.*
fillellipse(xc,yc,xrad,yrad)	*Fills up an ellipse with center (xc,yc), a=xrad & b=yrad.*	***fillellipse(320,240,100,50)****– fills an ellipse whose center is at (320,240) and the two axes at x=100 and y=50.*
drawpoly(x,y)	*Draws a polygon where x⇒number of points used to build the polygon and y⇒base address of the array containing the co-ordinate points.*	***drawpoly(4,array)*** *– draws a polygon containing 4 points and the base address is contained in the array array[].*
fillpoly(x,y)	*Fills up a polygon. x⇒number of points used to build the polygon and y⇒base address of the array containing the co-ordinate points.*	***fillpoly(x,y)*** *- draws a polygon containing 4 points and the base address is contained in the array array[]. It also fills up the polygon using the current fill style and color.*
getnewsettings(&vp)	*Filling up the elements of the structure viewporttype (ex vp) with the co-ordinates of the current viewport.*	
getpalette(&palette)	*Gets the set of color values. Palette is of structure palettetype.*	
setpalette(colornum,color)	*Changes the color values. setpalette changes the colormun entry in palette to color.*	***setpalette(0,5)*** *– Changes the first color in the current palette (background color) to actual color number 5.*
setallpalette(&palette)	*Changes all colors to original values.*	
kbhit()	*Returns true if keyboard is hit. This is useful for interactive graphics.*	
settextstyle(x,y,z)	*Sets font style. x⇒font type, y⇒font direction (e.g. VERT_DIR), z⇒point size.*	***settextstyle(DEFAULT_font,HORIZ_DIR,4)*** *– draws a text in the existing font and from left to right*

		and in 4 times the original size.
sound(x)	*Activates the speaker at a specific time unit in x Hz.*	***sound(7)*** *– will activate a sound at 7 Hz.*
nosound()	*Stops previously activated sound.*	
delay(x)	*Allowing the previously executed command to remain activated for a time unit x(in msec).*	***delay(6000)*** *– will make the command run for 6 sec.*
getimage(x,y,a,b,c)	*A function for storing the image. x⇒x co-ordinate of the top left corner of the block. y⇒y co-ordinate of the top left corner of the block. a⇒x co-ordinate of the bottom right corner of the block. b⇒y co-ordinate of the bottom right corner of the block. c⇒the address of the memory location from where the image would be stored.*	
putimage(x,y,c,d)	*Puts image on the screen. x⇒x coordinate of the top left corner. y⇒y coordinate of the top left corner. c⇒address of the memory from where the image has to be retrieved. XOR_PUT ⇒image was present in memory and screen, but resultant image on the screen will now be erased.*	
bar(left,top,right,bottom)	*Draws a filled-in two-dimensional rectangular bar. It does not draw the bar boundary. The rectangle is drawn using the current fill pattern and color.*	***bar(20,30,40,70)*** *– will draw a rectangle whose upper left corner is at (20,30) and lower right corner is at (40,70).*

With the above background, let us now have a look into a few examples of graphics programming in Turbo C/C++ environment. The program listings presented in this context cover a few topics from various chapters of this book. Once you are familiar with some of those topics, you can try out to compile/execute/modify some of these programs, to enhance your understanding.

Before presenting the program listings, we will first jot down some pertinent points regarding their usage. It will be convenient for you to use the programs after you read these general comments.

- All the programs presented in this appendix have been written in C and compiled and executed in Turbo C 3.0 compiler.
- It has been assumed that the user would give a correct set of inputs. For example, while inputting the coefficients of the plane of the equation in the normal form: lx + my + nz + d = 0, where l, m, n are the direction cosines of the normal direction of the specified plane, it is assumed that the user will supply valid values in the range 0 to 1, and satisfying $l^2 + m^2 + n^2 = 1$, and so on. Therefore, any rigorous checking for validity of the user-inputs has been omitted in order to avoid unnecessary complications in the programs.
- The inputs to four programs, namely, programs implementing polygon clipping, polygon filling, z-Buffer algorithm and PC surface generation, have been taken through files. The input data files for these programs are typically named as *.txt, where * represents the program name in which this input file is utilized. In each input file, multiple sets of inputs are prescribed. When the program is run the first set of inputs are used to produce the output. The other sets, if replaced in place of the first set, can be used as alternate data sets. The different data sets are separated by a blank line or ******. You could also provide your own input but there must be one to one correspondence between the required parameters and those supplied. The required parameters can easily be found by going through the input prompts in the programs. Moreover, in place of file inputs, console inputs can also be easily provided by commenting out the line where the file is opened using *fopen( )* and converting all the *fscanf (fp, ...)* to *scanf(....)*.
- All the programs have been made as generic as possible and never inputs have been taken inside the programs itself.
- It may be advisable to create a 'Common' folder that contains the utility functions (user defined header files), which are shared by many programs.
- In all 3D programs isometric projection is used to show the figure on the screen.
- The coordinate system has been converted to a right-handed one and all point are specified according to the right-handed co-ordinate system.
- All line drawings in all the programs have been done using Bresenhem's algorithm to avoid floating point calculations. The *round* function has been used to suitably round a coordinate when used in calculations with floating point variables.
- Effort has been taken to stick to ANSI standards as much as possible. However, BGI graphics functions themselves are not specified by ANSI and *clrscr( )* and *getch( )* functions have been used which according to ANSI standards are not portable.
- Variables have been named in lowercase with certain exceptions where the variable names are abbreviated forms such as AET_pointer, which stands for active edge table pointer, and so on.
- Function names start with uppercase (for example, *ScanFill* (...)).
- Often *char* have been used to store small integers to save space; especially in memory intensive programs like the ones implementing the z- Buffer algorithm.
- Global variables have been strictly avoided to write functions, which would be more generic, and also to avoid other problems typical to global variables.

- The path specified in the programs to include the user-defined header files assumes that the 'Programs' directory is placed in the C drive and the directory structure inside the 'Programs' directory is not changed. However, if this is not possible for some reason, then before running each program you need to change the drive name in the #include directives. For example, change `#include"C:\Programs\Graphics\Common\Project.h"` to `#include "D:\Programs\Graphics\Common\Project.h"`
- It is assumed in the programs that the Turbo C compiler is present in 'C:\tcc\' and accordingly the path is specified in the *initgraph*( ) function. If the Turbo C compiler is elsewhere, before running each program look for a function *InitGraph*(< string - path >) (user-defined) which passes the path to *initgraph*(…).. change the path inside *InitGraph*( ). For example, change: InitGraph("C:\\tcc\\bgi\\") to `InitGraph("D:\\TurboC3\\bgi\\");` if your Turbo C compiler is in the D drive with the name TurboC3 and so on.

Header files in C:\Programs\Graphics\Common

/*program Graphics.h*/

```
/* The commonly used functions in Graphics programs */

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


int InitGraph(char* path)
{
  int gd = DETECT, gm, errcode;

  initgraph(&gd,&gm,path);
  errcode = graphresult();

  if (graphresult() != grOk)
  {
    printf("Graphics error: %s\n", grapherrormsg(errcode));
    printf("Press any key to halt... ");
    getch();
    return EXIT_FAILURE;      /* terminate with an error code */
  }
  return EXIT_SUCCESS;
}
```

```
void CoordinateSystem(void)
{
  /* Draw the co-ordinate axes */

  line(0, getmaxy()/2, getmaxx(), getmaxy()/2);   /* x axis */
  line(getmaxx()/2, 0, getmaxx()/2, getmaxy());   /* y axis */

  outtextxy(  getmaxx()/2 - 10  ,  getmaxy()/2 - 10   , "O"  );
  outtextxy(  0                 ,  getmaxy()/2 - 10   , "X'" );
  outtextxy(  getmaxx() - 10    ,  getmaxy()/2 - 10   , "X"  );
  outtextxy(  getmaxx()/2 - 15  ,  0                  , "Y"  );
  outtextxy(  getmaxx()/2 - 15  ,  getmaxy() - 10     , "Y'" );
}

void PutPixel(int x, int y, int fill_color)
{
  /* Change to RH co-ordinate system & shift origin to centre */

  if( getmaxx()/2 + x <= getmaxx() && getmaxy()/2 - y <= getmaxy() )
    putpixel(getmaxx()/2 + x, getmaxy()/2 - y, fill_color);
}


void ColourMenu(void)
{
  printf("\nEnter the number corresponding to the colour of your choice(s) :\n\n");
  printf("\tÅÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅ\n");
  printf("\t³  BLACK         ³  0  ³\n");
  printf("\t³  BLUE          ³  1  ³\n");
  printf("\t³  GREEN         ³  2  ³\n");
  printf("\t³  CYAN          ³  3  ³\n");
  printf("\t³  RED           ³  4  ³\n");
  printf("\t³  MAGENTA       ³  5  ³\n");
  printf("\t³  BROWN         ³  6  ³\n");
  printf("\t³  LIGHTGRAY     ³  7  ³\n");
  printf("\t³  DARKGRAY      ³  8  ³\n");
  printf("\t³  LIGHTBLUE     ³  9  ³\n");
  printf("\t³  LIGHTGREEN    ³ 10  ³\n");
  printf("\t³  LIGHTCYAN     ³ 11  ³\n");
  printf("\t³  LIGHTRED      ³ 12  ³\n");
  printf("\t³  LIGHTMAGENTA  ³ 13  ³\n");
  printf("\t³  YELLOW        ³ 14  ³\n");
  printf("\t³  WHITE         ³ 15  ³\n");
  printf("\tÅÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÅ\n");
  printf("\n\t   YOUR CHOICE(S) : ");
}
```

/* program 2DTrnsfm.h*/

```
/* Functions for 2D transformations */

#include <math.h>

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

void Translate(int* x, int* y, int a, int b)
{
  *x = *x + a;
  *y = *y + b;
}

/* If Reflection is about x axis then p = y and so on... */

void ReflectAbtAxis(int* p)
{
  *p = -*p;
}

void RotateThruTheta(int* x, int* y, float theta)
{
  int tx, ty;
  tx = *x;    ty = *y;
  *x = round((tx * cos(theta)) - (ty * sin(theta)));
  *y = round((tx * sin(theta)) + (ty * cos(theta)));
}

void Scale(int* x, int* y, float alpha)
{
  *x = *x * alpha;
  *y = *y * alpha;
}
```

/* program 3DTrnsfm.h*/

```
/* Functions for 3D transformations */

#include <math.h>

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

void Translate(int* x, int* y, int * z, int a, int b, int c)
{
  *x = *x + a;
  *y = *y + b;
```

```
  *z = *z + c;
}

/* If Reflection is about x-y plane then m is z and so on... */

void ReflectAbtPlane(int* m)
{
  *m = -*m;
}

/* If Rotation is about x axis then p = y, q = z and so on... */

void RotateThruTheta(int* p, int* q, float theta)
{
  int tp, tq;
  tp = *p;    tq = *q;
  *p = round((tp * cos(theta)) - (tq * sin(theta)));
  *q = round((tp * sin(theta)) + (tq * cos(theta)));
}
```

/* program BresLine.h*/

```
/* Bresenham's Line Generation Algorithm */

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

#define INF 9999

void swap(int* a, int* b)
{
  int t;
  t = *a;   *a = *b;   *b = t;
}


void BresLine(int x1, int y1, int x2, int y2, int colour)
{
  int d, dy, dx, signdx, signdy;
  float m;

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

  if(dx == 0)
    m = INF;
  else
    m = (float) dy/dx;

  if( ( fabs(m) <= 1.0 && (x2 < x1) ) ||
```

```
    ( fabs(m) >  1.0 && (y2 < y1) ) )
  {
    swap(&x1, &x2);
    swap(&y1, &y2);
    dy = -dy;
    dx = -dx;
  }

  signdx = (dx == 0) ? 0 : abs(dx)/dx;
  signdy = (dy == 0) ? 0 : abs(dy)/dy;

  if(fabs(m) > 1.0)
    swap(&dx, &dy);

  PutPixel(x1, y1, colour);

  d =  2 * abs(dy) - abs(dx);

  do {
        if( d>=0 )
        {
          d += 2 * (abs(dy) - abs(dx));
          x1 += signdx;
          y1 += signdy;
        }
        else
        {
          d += 2 * abs(dy);
          if(fabs(m) <= 1.0)
            x1 += signdx;
          else
            y1 += signdy;
        }
        PutPixel(x1, y1, colour);
    } while( x1 != x2 || y1 != y2 );
}
```

/*program Project.h*/

```
#include <math.h>
#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

void Axonometric(int* x, int* y, int* z, int n, float phi, float theta)
{
  int tx, ty, tz, i;
  for(i=0;i<n;i++)
  {
    tx = x[i]; ty = y[i]; tz = z[i];
```

```
    x[i] = round((tx * cos(phi)) - (ty * sin(phi)));
    y[i] = round((tx * (sin(phi)) * cos(theta)) + (ty * (cos(phi)) * cos(theta)) - (tz *
sin(theta)));
    z[i] = 0;
  }
}
```

/*program ScanPoly.h*/

```
#define NULL 0
#define EMPTY -1
#define ODD 1
#define EVEN 0


typedef struct
        {
          int x1;
          int y1;
          int x2;
          int y2;
        } edge;

typedef struct active_edge
        {
          int ymax;
          float x;
          float dx;
          struct active_edge *next;
        } active_edge;


void swap(int* a, int* b);


void Sort(edge *ET, int sides)
{
  int i, j;
  edge temp[1];

  for(i=0; i<(sides - 1); i++)
    for(j=i+1; j<sides; j++)
    {
        if(ET[i].y1 > ET[j].y1)
        {
          temp[0] = ET[i];
          ET[i] = ET[j];
```

```
            ET[j] = temp[0];
          }
          if(ET[i].y1 == ET[j].y1)
            if(min(ET[i].x1, ET[i].x2) > min(ET[j].x1, ET[j].x2))
            {
              temp[0] = ET[i];
              ET[i] = ET[j];
              ET[j] = temp[0];
            }
    }
}


void DeleteEdge(active_edge *AET_pointer, int y)
{
  active_edge *edge = AET_pointer, *edge_next;

  if(edge->next != NULL)          /* AET is non empty */
  {
    while(AET_pointer->next->ymax == y)     /* 1st node deleted */
    {
        edge=AET_pointer->next;
        AET_pointer->next=AET_pointer->next->next;
        free(edge);
    }

    edge = AET_pointer->next;
    edge_next = edge->next;

    while(edge != NULL && edge->next != NULL)
    {
        while(edge_next->ymax != y)
        {
          edge = edge_next;
          if(edge->next != NULL)
            edge_next = edge_next->next;
          else
            break;
        }
        if(edge->next != NULL)
        {
          edge->next = edge_next->next;
          edge_next = edge_next->next;
          continue;
        }
    }
  }
}


void SwapEdge(active_edge *AET_pointer, int m, int n)
```

```
{
  active_edge *prev_1, *current_1, *prev_2, *current_2, *tmp_1, *tmp_2;
  int i = 2, j = 1;

                              /* Swapping nodes other than the first node */
  prev_1 = AET_pointer;
  current_1 = AET_pointer->next;

  while(i<m)
  {
    prev_1 = prev_1->next;
    current_1 = current_1->next;
    i++;
  }

  prev_2 = current_1;
  current_2 = current_1->next;

  while(j<(n-m))
  {
    prev_2 = prev_2->next;
    current_2 = current_2->next;
    j++;
  }

  tmp_1 = prev_1->next;
  prev_1->next = prev_2->next;
  tmp_2 = current_2->next;

  if ((n-m)==1)
  {
    current_2->next = tmp_1;     /* Swapping 2 adjacent nodes */
    current_1->next = tmp_2;
  }
  else
  {
    current_2->next = current_1->next;
    current_1->next = tmp_2;
    prev_2->next = tmp_1;
  }
}


void SortAET(active_edge *AET_pointer)
{
  int i, j;
  active_edge *edge = AET_pointer->next;
  active_edge *edge_next = edge->next;

  for(edge = AET_pointer->next, i = 2; edge->next != NULL; edge= edge->next, i++)
    for(edge_next = edge->next, j = i + 1; edge_next != NULL; j++)
```

```
    {
        if(edge->x  > edge_next->x)
        {
          SwapEdge(AET_pointer, i, j);
          edge = edge_next;             /* Update the pointers */
          edge_next = edge->next;
        }
        else
          edge_next = edge_next->next;   /* Not swapped, normal increment */
    }
}


void UpdateAET(active_edge *AET_pointer)
{
  active_edge *edge = AET_pointer->next;
  char parity = ODD;
  while(edge != NULL)
  {
    edge->x = (edge->x + edge->dx);
    edge = edge->next;
    parity = !parity;
  }
}



int SetupET(edge *ET, int sides)
{
  int i, j;
  for(i=0;i<sides;i++)
  {
    if(ET[i].y1 == ET[i].y2)      /* Eliminate Horizontal edges */
    {
        for(j=i+1;j<sides;j++)
          ET[j-1] = ET[j];
        sides--;
    }
    if(ET[i].y1 > ET[i].y2)       /* Sort to make y1 = ymin */
    {
        swap(&ET[i].x1,&ET[i].x2);
        swap(&ET[i].y1,&ET[i].y2);
    }
  }
  Sort(ET, sides);
  return sides;
}
```

Main program files

/* Program implementing Bresenham's Line Generation Algorithm */

```
#include <math.h>
#include "C:\Programs\Graphics\Common\Graphics.h"

# define INF 9999


int main()
{
  int x1, y1, x2, y2, colour;

  void BresLine(int x1, int y1, int x2, int y2, int colour);

  printf("\nEnter the co-ordinates of the first point.\n");
  scanf("%d %d", &x1, &y1);
  printf("\nEnter the co-ordinates of the second point.\n");
  scanf("%d %d", &x2, &y2);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi");
  CoordinateSystem();
  BresLine(x1, y1, x2, y2, colour);
  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void swap(int* a, int* b)
{
  int t;
  t = *a;   *a = *b;   *b = t;
}


void BresLine(int x1, int y1, int x2, int y2, int colour)
{
  int d, dy, dx, signdx, signdy;
  float m;

  void swap(int* a, int* b);

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

```
  if(dx == 0)
    m = INF;
  else
    m = (float) dy/dx;

  if( ( fabs(m) <= 1.0 && (x2 < x1) ) ||
      ( fabs(m) >  1.0 && (y2 < y1) ) )
  {
    swap(&x1, &x2);
    swap(&y1, &y2);
    dy = -dy;
    dx = -dx;
  }

  signdx = (dx == 0) ? 0 : abs(dx)/dx;
  signdy = (dy == 0) ? 0 : abs(dy)/dy;

  if(fabs(m) > 1.0)
    swap(&dx, &dy);

  PutPixel(x1, y1, colour);

  d =  2 * abs(dy) - abs(dx);

  do {
          if( d>=0 )
          {
            d += 2 * (abs(dy) - abs(dx));
            x1 += signdx;
            y1 += signdy;
          }
          else
          {
            d += 2 * abs(dy);
            if(fabs(m) <= 1.0)
              x1 += signdx;
            else
              y1 += signdy;
          }
          PutPixel(x1, y1, colour);
    } while( x1 != x2 || y1 != y2 );
}
```

/* Program for Bresenham's Circle Generation Algorithm */

```
#include "C:\Programs\Graphics\Common\Graphics.h"


int main()
{
  int xc, yc, r, colour;

  void BresCrcl(int xc, int yc, int r, int colour);

  printf("\nEnter the co-ordinates of the centre of the circle (x,y) : \n");
  scanf("%d %d", &xc, &yc);
  printf("\nEnter the radius of the circle : \n");
  scanf("%d", &r);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi");
  CoordinateSystem();
  BresCrcl(xc, yc, r, colour);
  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void BresCrcl(int xc, int yc, int r, int colour)
{
  int x, y, d;
  x = 0;  y = r;  d = 3 - 2*r;
  while( x <= y )
  {
    PutPixel( xc + x, yc + y, colour );
    PutPixel( xc + x, yc - y, colour );
    PutPixel( xc - x, yc + y, colour );
    PutPixel( xc - x, yc - y, colour );
    PutPixel( xc + y, yc + x, colour );
    PutPixel( xc + y, yc - x, colour );
    PutPixel( xc - y, yc + x, colour );
    PutPixel( xc - y, yc - x, colour );

    if( d >= 0 )
    {
        d = d + 4*(x-y) + 10;
        y--;
    }
    else
        d = d +4*x + 6;
```

```
    x++;
  }
}
```

/* Ellipse Generation using 4 way Symmetry */

```
/* Issues :  It is neccessary to make the variables a, b, x, y as doubles
             though they themselves do not take large values because they
             appear in expressions which often take large values...

             e.g.  while( (2*b*b*x) < (2*a*a*y) ) .... will give wrong
             results even if the value of a & b are as small as 60, 20 etc.

             even if only x & y are made double and we let the automatic
             type conversions do the rest then too the terms in
             calculation of ds which involve only a & b will give wrong
             results.

             double was neccessary even if a, b, x, y always assume int
             values...as it doesn't cover the larger of the drawable
             ellipses.
*/


#include "C:\Programs\Graphics\Common\Graphics.h"


int main()
{
  int xc, yc, colour;
  double a, b;

  void EllipseAtOrigin(int xc, int yc, double a, double b, int colour);

  printf("\nEnter the co-ordinates of the centre of the ellipse (x,y) : \n");
  scanf("%d %d", &xc, &yc);
  printf("\nEnter the major and minor radii : \n");
  scanf("%lf %lf", &a, &b);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi");
  CoordinateSystem();
  EllipseAtOrigin(xc, yc, a, b, colour);
  getch();
```

```
  closegraph();
  return EXIT_SUCCESS;
}



void EllipseAtOrigin(int xc, int yc, double a, double b, int colour)
{
  double x = 0, y = b;
  double d1, d2;

  void PlotEllipse(double x,double y, int xc, int yc, int colour);

  d1 = (b*b) - (a*a*b) + (0.25*a*a);
  PlotEllipse(x, y, xc, yc, colour);

  while( (2*b*b*x) < (2*a*a*y) )
  {
    /* Region 1 */

    if(d1 < 0)   /* Select E */
    {
        d1 += (b*b)*(2*x + 3);
        x++;
    }
    else        /* Select SE */
    {
        d1 += (b*b)*(2*x + 3) + (a*a)*(-2*y + 2);
        y--;
        x++;
    }
    PlotEllipse(x, y, xc, yc, colour);
  }

  d2 = (b*b)*(x + 0.5)*(x + 0.5) + (a*a)*(y -1)*(y - 1) - (a*a)*(b*b);

  while(y > 0)
  {
    /* Region 2 */

    if(d2 < 0)   /* Select SE */
    {
        d2 += (b*b)*(2*x + 2) + (a*a)*(-2*y + 3);
        x++;
        y--;
    }
    else        /* Select S */
    {
        d2 += (a*a)*(-2*y + 3);
        y--;
```

```
    }
    PlotEllipse(x, y, xc, yc, colour);
  }
}


void PlotEllipse(double x,double y, int xc, int yc, int colour)
{
    PutPixel( xc + x, yc + y, colour );
    PutPixel( xc + x, yc - y, colour );
    PutPixel( xc - x, yc + y, colour );
    PutPixel( xc - x, yc - y, colour );
}
```

/* Program to reflect a triangle w.r.t. a given straight line */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\2DTrnsfm.h"


int main()
{
  int i;
  int c, x[3], y[3], colour;
  float m;

  void Reflect(float m, int c, int* x, int* y, int colour);

  printf("Enter the slope and the intercept(c) of the reflecting line : ");
  scanf("%f %d", &m, &c);
  printf("Enter the co ordinates of the 3 vertices of the triangle : ");
  for(i=0;i<3;i++)
    scanf("%d %d", &x[i], &y[i]);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi\\");
  CoordinateSystem();
  Reflect(m, c, x, y, colour);
```

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


void Reflect(float m, int c, int* x, int* y, int colour)
{
  int i;
  int my1, my2;
  int maxx = getmaxx()/2;
  do
  {
    my1 = (-m * maxx-- + c);
    my2 = ( m * maxx-- + c);
  } while(my1 > getmaxx()/2 || my2 > getmaxx()/2);

  BresLine( -maxx++, my1, maxx, my2, colour);

  for(i=0;i<3;i++)
    BresLine(x[i], y[i], x[(i+1)%3], y[(i+1)%3], colour);

  for(i=0;i<3;i++)
  {
    Translate(&x[i], &y[i], 0, -c);
    RotateThruTheta(&x[i], &y[i], -atan(m));
    ReflectAbtAxis(&y[i]);
    RotateThruTheta(&x[i], &y[i], atan(m));
    Translate(&x[i], &y[i], 0, c);
  }

  for(i=0;i<3;i++)
    BresLine(x[i], y[i], x[(i+1)%3], y[(i+1)%3], colour);
}
```

/* Program to rotate a triangle w.r.t. a given point */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\2DTrnsfm.h"

#define PI 3.141592654


int main()
{
  int a, b;
  int i, x[3], y[3], colour;
  float theta;
```

```
  void Rotate(int a, int b, int* x, int* y, float theta, int colour);

  printf("Enter the co-ordinates of the point of rotation : ");
  scanf("%d %d", &a, &b);
  printf("Enter the angle of rotation in degrees : ");
  scanf("%f", &theta);
  printf("Enter the co-ordinates of the 3 vertices of the triangle : ");
  for(i=0;i<3;i++)
    scanf("%d %d", &x[i], &y[i]);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi\\");
  CoordinateSystem();
  Rotate(a, b, x, y, theta, colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void Rotate(int a, int b, int* x, int* y, float theta, int colour)
{
  int i;
  PutPixel(a, b, WHITE);

  theta = (PI/180)*theta;

  for(i=0;i<3;i++)
    BresLine(x[i], y[i], x[(i+1)%3], y[(i+1)%3], colour);

  for(i=0;i<3;i++)
  {
    Translate(&x[i], &y[i], -a, -b);
    RotateThruTheta(&x[i], &y[i], theta);
    Translate(&x[i], &y[i], a, b);
  }

  for(i=0;i<3;i++)
    BresLine(x[i], y[i], x[(i+1)%3], y[(i+1)%3], colour);
}
```

/* Program for scaling a polygon w.r.t. a given point */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\2DTrnsfm.h"


int main()
{
  int i, n, a, b, colour;
  int *x, *y;
  float alpha;

  void ScalePoly(int a, int b, float alpha, int n, int* x, int* y, int colour);

  printf("Enter the nos. of sides of the polygon : ");
  scanf("%d", &n);
  printf("Enter the scaling factor: ");
  scanf("%f", &alpha);
  printf("Enter the co-ordinates of the centre of scaling : ");
  scanf("%d %d", &a, &b);

  x = (int*) malloc( n * sizeof(int) );
  y = (int*) malloc( n * sizeof(int) );

  printf("Enter the co-ordinates of the polygon : ");
  for(i=0;i<n;i++)
    scanf("%d %d", &x[i], &y[i]);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tc\\bgi\\");
  CoordinateSystem();
  ScalePoly(a, b, alpha, n, x, y, colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void ScalePoly(int a, int b, float alpha, int n, int* x, int* y, int colour)
{
  int i;
  PutPixel(a, b, WHITE);

  for(i=0;i<n;i++)
    BresLine(x[i], y[i], x[(i+1)%n], y[(i+1)%n], colour);
```

```
  for(i=0;i<n;i++)
  {
    Translate(&x[i], &y[i], -a, -b);
    Scale(&x[i], &y[i], alpha);
    Translate(&x[i], &y[i], a, b);
  }
  for(i=0;i<n;i++)
    BresLine(x[i], y[i], x[(i+1)%n], y[(i+1)%n], colour);
}
```

/* Program for Isometric Projection of any parallelopiped */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\Project.h"

#define PI 3.141592653
#define PHI (PI/180) * 45
#define THETA (PI/180) * 54.7


int main()
{
  int i, colour;
  int x[8], y[8], z[8];
  int xmin, xmax, ymin, ymax, zmin, zmax;

  void GetMaxMin(int* arr, int count, int* min, int* max);

  printf("Enter 3 vertices of a parallelopiped such that it is uniquely defined : \n");
  for(i=0;i<3;i++)
    scanf("%d %d %d", &x[i], &y[i], &z[i]);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  GetMaxMin(x, 3, &xmin, &xmax);
  GetMaxMin(y, 3, &ymin, &ymax);
  GetMaxMin(z, 3, &zmin, &zmax);

  /* Generate the vertices of the paralellopiped */

  x[0] = x[1] = x[2] = x[3] = xmin;
  x[4] = x[5] = x[6] = x[7] = xmin + (xmax - xmin);

  y[0] = y[3] = y[4] = y[7] = ymin;
  y[1] = y[2] = y[5] = y[6] = ymin + (ymax - ymin);
```

```
  z[0] = z[1] = z[4] = z[5] = zmin;
  z[2] = z[3] = z[6] = z[7] = zmin + (zmax - zmin);

  Axonometric(x, y, z, 8, PHI, THETA);   /* Isometric Projection */

  InitGraph("C:\\tcc\\bin\\");
  CoordinateSystem();

  for(i=0;i<4;i++)
    BresLine(x[i], y[i], x[(i+1)%4], y[(i+1)%4], colour);

  for(;i<7;i++)
    if(i!=5 && i!=6)
        BresLine(x[i], y[i], x[(i+1)], y[(i+1)], colour);

  BresLine(x[7], y[7], x[4], y[4], colour);

  for(i=0;i<4;i++)
    if(i!=2)
        BresLine(x[i], y[i], x[(i+4)], y[(i+4)], colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void GetMaxMin(int* arr, int count, int* min, int* max)
{
  int i;
  *max = *min = arr[0];
  for(i=1;i<count;i++)
  {
    if(arr[i] < *min)
        *min = arr[i];
    if(arr[i] > *max)
        *max = arr[i];
  }
}
```

/* Program for rotating a polygon w.r.t. a given line by a specified angle and generating the co-ordinates of the rotated polygon in 3D space and draw

Isometric Projections of original as well as the rotated polygon. */

```
#include <stdlib.h>

#include "C:\Programs\Graphics\Common\3DTrnsfm.h"
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\Project.h"

#define PI 3.141592654
#define PHI (PI/180) * 45
#define THETA (PI/180) * 54.7


int main()
{
  int i, sides, colour;
  int *xpoly, *ypoly, *zpoly;
  int *copy_xpoly, *copy_ypoly, *copy_zpoly;
  int xline[2], yline[2], zline[2];
  float theta;

  void RotatePolygon(int* xline, int* yline, int* zline, int* xpoly, int* ypoly, int* zpoly, int
sides, float theta);
  clrscr();

  printf("Enter the co ordinates of the line of rotation : ");
  for(i=0;i<2;i++)
    scanf("%d %d %d", &xline[i], &yline[i], &zline[i]);

  printf("Enter the nos. of sides in the polygon : ");
  scanf("%d", &sides);

  xpoly = (int*) malloc(sides * sizeof(int));
  ypoly = (int*) malloc(sides * sizeof(int));
  zpoly = (int*) malloc(sides * sizeof(int));
  copy_xpoly = (int*) malloc(sides * sizeof(int));
  copy_ypoly = (int*) malloc(sides * sizeof(int));
  copy_zpoly = (int*) malloc(sides * sizeof(int));

  printf("Enter the co ordinates of the polygon in sequence : ");
  for(i=0;i<sides;i++)
  {
    scanf("%d %d %d", &xpoly[i], &ypoly[i], &zpoly[i]);
    copy_xpoly[i] = xpoly[i];
    copy_ypoly[i] = ypoly[i];
    copy_zpoly[i] = zpoly[i];
  }

  printf("Enter the angle of rotation in degrees : ");
  scanf("%f", &theta);
```

```
  theta = (PI/180) * theta;

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tcc\bgi\\");

  printf("\nIsometric Projection of the original polygon...");
  CoordinateSystem();

  Axonometric(copy_xpoly, copy_ypoly, copy_zpoly, sides, PHI, THETA);  /* Isometric */

  for(i=0;i<sides;i++)
    BresLine(copy_xpoly[i],          copy_ypoly[i],          copy_xpoly[(i+1)%sides],
copy_ypoly[(i+1)%sides], colour);

  free(copy_xpoly);
  free(copy_ypoly);
  free(copy_zpoly);

  printf("  Press any key to rotate...\r");
  getch();

  RotatePolygon(xline, yline, zline, xpoly, ypoly, zpoly, sides, theta);

  cleardevice();

  printf("The co ordinates of the polygon after rotation in sequence :\n\n");
  for(i=0;i<sides;i++)
    printf("(%d, %d, %d)\t", xpoly[i], ypoly[i], zpoly[i]);

  printf("\n\nIsometric Projection of the rotated polygon...");
  CoordinateSystem();
  Axonometric(xpoly, ypoly, zpoly, sides, PHI, THETA);  /* Isometric */

  for(i=0;i<sides;i++)
    BresLine(xpoly[i], ypoly[i], xpoly[(i+1)%sides], ypoly[(i+1)%sides], colour);

  free(xpoly);
  free(ypoly);
  free(zpoly);

  getch();
  return EXIT_SUCCESS;
}


void RotatePolygon(int* xline, int* yline, int* zline, int* xpoly, int* ypoly, int* zpoly, int
sides, float theta)
{
```

```
  int i;
  int a = xline[0], b = yline[0], c = zline[0];
  float sinetheta_1, sinetheta_2;

  for(i=0;i<sides;i++)
    Translate(&xline[i], &yline[i], &zline[i], -a, -b, -c);

  sinetheta_1 = (float)yline[1]/(sqrt((float)yline[1]*yline[1] + (float)zline[1]*zline[1]));
  sinetheta_2 = (float)xline[1]/(sqrt((float)xline[1]*xline[1] + (float)yline[1]*yline[1] +
(float)zline[1]*zline[1]));

  for(i=0;i<sides;i++)
  {
    RotateThruTheta(&ypoly[i], &zpoly[i], +asin(sinetheta_1));
    RotateThruTheta(&zpoly[i], &xpoly[i], -asin(sinetheta_2));
    RotateThruTheta(&xpoly[i], &ypoly[i], theta);
    RotateThruTheta(&zpoly[i], &xpoly[i], +asin(sinetheta_2));
    RotateThruTheta(&ypoly[i], &zpoly[i], -asin(sinetheta_1));
    Translate(&xline[i], &yline[i], &zline[i], +a, +b, +c);
  }
}
```

/* Cohen Sutherland Line Clipping Algorithm Program*/

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"

#define GetBit(x,p) (x >> p) & ~(~0 << 1)     /* Get bit number p from x */

int xmin, ymin, xmax, ymax;


int main()
{
  int x1, y1, x2, y2;
  int window_colour, line_colour;

  void ClipLine(int x1, int y1, int x2, int y2, int window_colour, int line_colour);

  printf("Enter the co ordinates of the line to be clipped (x1,y1) (x2, y2) : ");
  scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
  printf("Enter the co ordinates of the viewing window (xmin, ymin) (xmax, ymax) : ");
  scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);

  if( (xmin > xmax) || (ymin > ymax) )
  {
    printf("Enter the co ordinates of the viewport corretly.\n");
```

```
    getch();
    return EXIT_FAILURE;
  }

  printf("Enter the colour to be used for drawing the window and the line respectively :
\n");
  ColourMenu();
  scanf("%d %d", &window_colour, &line_colour);


  InitGraph("C:\\tc\\bgi\\");
  CoordinateSystem();
  ClipLine(x1, y1, x2, y2, window_colour, line_colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void ClipLine(int x1, int y1, int x2, int y2, int window_colour, int line_colour)
{
  int x, y, i;
  char code1, code2;
  static int point1[2], point2[2];   /* Must be initialised to 0 for      */
                                     /* lines which though not clipping   */
                                     /* candidates give code1 & code2 = 0 */

  char GetCode(int x, int y);
  void DrawViewport(int colour);
  void GetPoint(char code, int x1, int y1, int x2, int y2, int* point, int pointnos);

  BresLine(x1, y1, x2, y2, line_colour);         /* Draw the Line */
  DrawViewport(window_colour);

  code1 = GetCode(x1, y1);
  code2 = GetCode(x2, y2);

  printf("Press any key to clip the line...");
  getch();

  cleardevice();
  CoordinateSystem();
  DrawViewport(window_colour);
  printf("\rThe Clipped line and the Viewport.");

  if(!(code1 & code2))             /* Line is a clipping candidate */
  {
    GetPoint(code1, x1, y1, x2, y2, point1, 1);
    GetPoint(code2, x1, y1, x2, y2, point2, 2);
    BresLine(point1[0], point1[1], point2[0], point2[1], line_colour);  /* Clipped Line */
```

```
  }
}


char GetCode(int x, int y)
{
  char code = 0;

  code =              (char)((y - ymax) > 0 ? 1 : 0);   /* Bit 3   MSB */
  code = (code << 1) | (char)((ymin - y) > 0 ? 1 : 0);   /* Bit 2       */
  code = (code << 1) | (char)((x - xmax) > 0 ? 1 : 0);   /* Bit 1       */
  code = (code << 1) | (char)((xmin - x) > 0 ? 1 : 0);   /* Bit 0   LSB */
  return code;
}


void DrawViewport(int colour)
{
  BresLine(xmin, ymin, xmin, ymax, colour);  /* Draw the Viewing Window */
  BresLine(xmin, ymax, xmax, ymax, colour);
  BresLine(xmax, ymax, xmax, ymin, colour);
  BresLine(xmax, ymin, xmin, ymin, colour);
}


void GetPoint(char code, int x1, int y1, int x2, int y2, int* point, int pointnos)
{
  int i, x, y;
  if((code & code) == 0)       /* If the endpoint is within the viewport */
  {
    point[0] = (pointnos == 1) ? x1 : x2;
    point[1] = (pointnos == 1) ? y1 : y2;
    return;
  }
  else
  {
    for(i=3;i>=0;i--)
    {
        if( GetBit(code, i) )
        {
          if(i==3 || i==2)
          {
            y = i==3 ? ymax : ymin;
            x = (((y - y1) * (x2 - x1))/(y2 - y1)) + x1;
            if(x <= xmax && x >= xmin)
            {
                point[0] = x;
                point[1] = y;
                break;
            }
```

```
            }

            if(i==1 || i==0)
            {
              x = i==1 ? xmax : xmin;
              y = (((y2 - y1) * (x - x1))/(x2 - x1)) + y1;
              if(y <= ymax && y >= ymin)
              {
                    point[0] = x;
                    point[1] = y;
                    break;
              }
            }
          }
    }
  }
}
```

/* Polygon Clipping Program using Sutherland Hodgman Algorithm */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

#define INFINITY 9999
#define OUTSIDE 0
#define INSIDE 1

FILE *fp;

typedef struct vertexlist
        {
          int x;
          int y;
          struct vertexlist *next;
        } vertex;


int main()
{
  vertex *window, *polygon, *clipped_poly;
```

```
  int window_colour, polygon_colour;

  void CreateVertexList(vertex *v);
  void Draw(vertex *v, int colour);
  vertex* PolyClip(vertex *window, vertex *polygon);

  fp = fopen("C:\\Programs\\Graphics\\Input\\ClipPoly.txt", "rt");
  clrscr();

  window = (vertex*) malloc(sizeof(vertex));
  polygon = (vertex*) malloc(sizeof(vertex));
  clipped_poly = (vertex*) malloc(sizeof(vertex));

  printf("Enter the vertices of the cilpping polygon anticlockwise : \n");
  CreateVertexList (window);

  printf("Enter the vertices of the polygon to be clipped anticlockwise : \n");
  CreateVertexList (polygon);

  printf("\nEnter the colour to be used for drawing the window and the polygon
respectively.");
  ColourMenu();
  scanf("%d %d", &window_colour, &polygon_colour);

  InitGraph("C:\\tcc\\bgi\\");
  CoordinateSystem();

  Draw(window, window_colour);
  Draw(polygon, polygon_colour);

  printf("Press any Key to Clip the Polygon...");
  getch();

  clipped_poly = PolyClip(window, polygon);

  cleardevice();
  CoordinateSystem();

  Draw(clipped_poly, polygon_colour);

  free(clipped_poly);
  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void CreateVertexList(vertex *v)
{
  vertex *first_vertex = v;
  char more = 'y';
```

```
  while(more == 'y')
  {
    fscanf(fp, "%d %d",&v->x,&v->y);
    printf("Insert another vertex ? (y/n) : ");
    fflush(stdin);
    fscanf(fp, "%c", &more);
    v->next = (vertex*) malloc(sizeof(vertex));
    if(more == 'y')
        v = v->next;
    else
    {                           /* The last & the first vertex */
        v = v->next;            /* are made same to close the polygons */
        v->x = first_vertex->x;
        v->y = first_vertex->y;
        v->next = NULL;
    }
  }
}


void Draw(vertex *v, int colour)
{
  vertex *point;
  for(point = v ; point->next != NULL ; point = point->next )
    BresLine(point->x, point->y, point->next->x, point->next->y, colour);
}



vertex* PolyClip(vertex *window, vertex *polygon)
{
  vertex *p, *w, *i;
  static vertex *VOL_head;
  int prev_x, prev_y;
  char prev_vertex_status;

  int IsInside(int x, int y, int x1, int y1, int x2, int y2);
  void AppendToVOL(vertex *p, int x, int y);
  vertex* IntersectPoint(int x1, int y1, vertex *p, vertex *w);

  VOL_head = (vertex*) malloc(sizeof(vertex));
  VOL_head->next = NULL;     /* The first node of VOL has no valid data */
  p = polygon;
  w = window;

  if( IsInside(p->x, p->y, w->x, w->y, w->next->x, w->next->y) )
  {
    prev_vertex_status = INSIDE;
    AppendToVOL(VOL_head, p->x, p->y);
  }
```

```
  else
    prev_vertex_status = OUTSIDE;

  prev_x = p->x;
  prev_y = p->y;
  p = p->next;

  while(p != NULL)
  {
    if( IsInside(p->x, p->y, w->x, w->y, w->next->x, w->next->y) )
    {
          if(prev_vertex_status == OUTSIDE)
          {
            i = IntersectPoint(prev_x, prev_y, p, w);
            AppendToVOL(VOL_head, i->x, i->y);
          }
          AppendToVOL(VOL_head, p->x, p->y);
          prev_vertex_status = INSIDE;
    }
    else
          if(prev_vertex_status == INSIDE)
          {
            i = IntersectPoint(prev_x, prev_y, p, w);
            AppendToVOL(VOL_head, i->x, i->y);
            prev_vertex_status = OUTSIDE;
          }
    prev_x = p->x;
    prev_y = p->y;
    p = p->next;
  }
            /* Closing the currently clipped Polygon */
  AppendToVOL(VOL_head, VOL_head->next->x, VOL_head->next->y);

  free(polygon);

  if(w->next->next != NULL)
    PolyClip(window->next, VOL_head->next);

  free(window);
  return  VOL_head->next;
}



int IsInside(int x, int y, int x1, int y1, int x2, int y2)
{
  if( (long)(x2 - x1) * (y - y1) >= (long)(y2 - y1) * (x - x1) )
    return INSIDE;
  else
    return OUTSIDE;
}
```

```
void AppendToVOL(vertex *p, int x, int y)
{
  vertex *new_vertex;
  new_vertex = (vertex*) malloc(sizeof(vertex));
  while(p->next != NULL)
    p = p->next;
  p->next = new_vertex;     /* Closing the recentmost clipped polygon */
  new_vertex->x = x;
  new_vertex->y = y;
  new_vertex->next = NULL;
}


vertex* IntersectPoint(int x1, int y1, vertex *p, vertex *w)
{
  vertex *i;
  int x2, y2, x3, y3, x4, y4;
  float m1, m2;

  i = (vertex*) malloc(sizeof(vertex));

  x2 = p->x ; y2 = p->y;
  x3 = w->x ; y3 = w->y;
  x4 = w->next->x ; y4 = w->next->y;

  m1 = (x1 == x2) ? INFINITY : (y1 - y2)/(x1 - x2);
  m2 = (x3 == x4) ? INFINITY : (y3 - y4)/(x3 - x4);

  if(m1 == INFINITY)
  {
    i->x = x1;
    i->y = round(m2 * (i->x - x3) + y3);
  }
  else
  {
    if(m2 == INFINITY)
        i->x = x3;
    else
        i->x = round(((y3 - (m2 * x3)) - (y1 - (m1 * x1)))/(m1 - m2));
    i->y = round(m1 * (i->x - x1) + y1);
  }
  return i;
}
```

/* Program for Filling a Polygon by Scan-Line Algorithm */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
#include "C:\Programs\Graphics\Common\ScanPoly.h"

edge *ET;      /* Edge Table */


int main()
{
  int i, j;
  int sides, ymin, fill_colour;
  FILE *fp;

  void FillPoly(int sides, int ymin, int fill_colour);
  clrscr();

  fp = fopen("C:\\Programs\\Graphics\\Input\\PolyFill.txt","rt");

  printf("Enter the nos. of edges in the polygon : ");
  fscanf(fp, "%d", &sides);

  ET = (edge*) malloc(sides * sizeof(edge));

  printf("Enter the vertices of the polygon in sequence : ");
  fscanf(fp, "%d %d", &ET[0].x1, &ET[0].y1);

  for(i=1;i<sides;i++)
  {
    fscanf(fp, "%d %d", &ET[i].x1, &ET[i].y1);
    ET[i-1].x2 =  ET[i].x1;
    ET[i-1].y2 =  ET[i].y1;
  }
  ET[i-1].x2 =  ET[0].x1;          /* Closing the Polygon */
  ET[i-1].y2 =  ET[0].y1;

  printf("Enter the colour to be used for filling : \n");
  ColourMenu();
  scanf("%d", &fill_colour);

  InitGraph("C:\\tcc\\bgi\\");
  CoordinateSystem();

  printf("\nPress any key to fill...\r");

  for(i=0;i<sides;i++)
    BresLine(ET[i].x1, ET[i].y1, ET[i].x2, ET[i].y2, BLUE);
```

```
  getch();

  sides = SetupET(ET, sides);

  cleardevice();
  CoordinateSystem();

  FillPoly(sides, ET[0].y1, fill_colour);
  free(ET);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void FillPoly(int sides, int ymin, int fill_colour)
{
  int y = ymin;
  int ET_pointer = 0;

  void ScanFill(active_edge *AET_pointer, int y, int fill_colour);

  /* Initialise Active Edge Table */
  active_edge *AET_pointer = (active_edge*) malloc(sizeof(active_edge)) ;
  active_edge *e, *new_edge;

  AET_pointer->next = NULL;

  while (ET_pointer != EMPTY || AET_pointer->next != NULL)
  {
    DeleteEdge(AET_pointer, y);

    while(ET_pointer != EMPTY && ET[ET_pointer].y1 == y)
    {
        e = AET_pointer;
        while(e->next != NULL)
          e = e->next;
        new_edge = (active_edge*) malloc(sizeof(active_edge));
        new_edge->ymax = ET[ET_pointer].y2;
        new_edge->x = ET[ET_pointer].x1;
        new_edge->dx    =    (float)(ET[ET_pointer].x2    -    ET[ET_pointer].x1)    /
(ET[ET_pointer].y2 - ET[ET_pointer].y1);
        e->next = new_edge;
        new_edge->next = NULL;
        if(ET_pointer == (sides -1))
          ET_pointer = EMPTY;
        else
          ET_pointer++;
    }
```

```
    if(AET_pointer->next != NULL)        /* If AET is non empty */
    {
        SortAET(AET_pointer);
        ScanFill(AET_pointer, y, fill_colour);

        y++;
        UpdateAET(AET_pointer);
    }
  }
  free(e);
  free(new_edge);
  free(AET_pointer);
}



void ScanFill(active_edge *AET_pointer, int y, int fill_colour)
{
  active_edge *edge = AET_pointer->next;
  active_edge *edge_next = edge->next;
  char parity = ODD;

  while(edge_next != NULL)
  {
    if(parity == ODD)
        BresLine(ceil(edge->x), y, floor(edge_next->x), y, fill_colour);
    edge = edge_next;
    edge_next = edge_next->next;
    parity = !parity;
  }
}
```

/* Z-Buffer Algorithm */

/* Since Turbo C 3.0 under MS DOS doesn't allow Z buffer and Frame Buffer of the size [640][480], the scan conversion has been done in strips. After scanning a polygon for a strip is completed and before moving on to the next polygon, the status of the polygon is kept in a structure called **status** so that next time the same polygon is taken up for the next strip the status is restored, and thereby AET_pointer, ET_pointer and the scan line currently to be processed is no lost. The same is done for every polygon. Thus this program works correctly for any polygon size as long as the polygon remains within the screen area. */

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

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\ScanPoly.h"

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

#define INFINITY 9999
#define MAX_X 640
#define MAX_Y 1
#define LOOP ((getmaxy() + 1) / MAX_Y)

#define map_x(x) (x + getmaxx()/2)
#define map_y(y) (y + getmaxy()/2 - loop_count * MAX_Y)
#define revmap_y(y) (getmaxy() - y - (loop_count * MAX_Y))

int Z[MAX_Y][MAX_X];        /* z-Buffer */
char F[MAX_Y][MAX_X];       /* Frame Buffer */

edge **ET;              /* Edge Table Array */

int loop_count;
float *l, *m, *n, *d;

typedef struct
        {
          int y;
          int ET_pointer;
          active_edge *AET_pointer;
        } status;


int main()
{
  status *poly_status;
  int ymax;
  unsigned int i, j, p, q;
  char *initialised;
  unsigned char *sides, polygons;
  unsigned char *fill_colour, back_colour;
  unsigned char init_flag;
  FILE *fp;

  void FillPoly(int sides, int ymin, int fill_colour, int ymax, int poly_nos, int init_flag, status
*poly_status, int polygons);
  clrscr();

  fp = fopen("C:\\Programs\\Graphics\\Input\\z-Buffer.txt", "rt");
  printf("Set the Background colour : \n");
  ColourMenu();
  fscanf(fp, "%d", &back_colour);
```

```
  printf("Enter the nos. of polygons : ");
  fscanf(fp, "%d", &polygons);

  ET = (edge **) malloc(polygons * sizeof(edge *));

  l = (float *) malloc(polygons * sizeof(float));
  m = (float *) malloc(polygons * sizeof(float));
  n = (float *) malloc(polygons * sizeof(float));
  d = (float *) malloc(polygons * sizeof(float));

  sides = (char *) malloc(polygons * sizeof(char));
  fill_colour = (char *) malloc(polygons * sizeof(char));
  poly_status = (status *) malloc(polygons * sizeof(status));

  initialised = (char *) malloc(polygons * sizeof(char));
  for(i = 0; i < polygons;i++)
    initialised[i] = -1;

  if(l == NULL || m == NULL || n == NULL || d == NULL || ET == NULL || sides == NULL
|| initialised == NULL || fill_colour == NULL || poly_status == NULL)
  {
    printf("Memory Allocation unsuccessful.\n");
    getch();
    return EXIT_FAILURE;
  }

  printf("Assume that the polygons are planar with eqn : lx + my +nz + d = 0\n");

  for(i = 0; i< polygons; i++)
  {
    printf("Enter the coefficients of the equation of the plane of the polygon %d : \n(l, m,
n, d) : ", i + 1);
    fscanf(fp, "%f %f %f %f", &l[i], &m[i], &n[i], &d[i]);

    printf("Enter the nos. of sides in polygon %d : ", i + 1);
    fscanf(fp, "%d", &sides[i]);

    ET[i] = (edge *) malloc(sides[i] * sizeof(edge));
    if(ET[i] == NULL)
    {
        printf("Memory Allocation unsuccessful.\n");
        getch();
        return EXIT_FAILURE;
    }

    printf("Enter the x and y co-ordinates of polygon %d in sequence.\n", i + 1);
    printf("The z co-ordinate for each vertex will be determined through the equation \nof
the plane : ");
    for(j = 0; j < sides[i]; j++)
    {
        fscanf(fp, "%d %d", &ET[i][j].x1, &ET[i][j].y1);
```

```
            ET[i][j-1].x2 =  ET[i][j].x1;
            ET[i][j-1].y2 =  ET[i][j].y1;
        }
        ET[i][j-1].x2 =  ET[i][0].x1;          /* Closing the Polygon */
        ET[i][j-1].y2 =  ET[i][0].y1;

        printf("Enter the colour to be used for filling : \n");
        ColourMenu();
        fscanf(fp, "%d", &fill_colour[i]);

        sides[i] = SetupET(ET[i], sides[i]);
    }

    InitGraph("C:\\tcc\\bgi\\");

    for(loop_count = 1; loop_count <= LOOP; loop_count++)
    {
        for(i = 0; i < MAX_Y; i++)
            for(j = 0; j < MAX_X; j++)
            {
                Z[i][j] = INFINITY;
                F[i][j] = back_colour;
            }
        ymax = -getmaxy()/2 + (loop_count * MAX_Y);
        for(i = 0; i < polygons; i++)
        {
            if( ET[i][0].y1 <= ymax )
            {
                if(initialised[i] == 1)
                    init_flag = 1;
                else
                {
                    init_flag = 0;
                    initialised[i] = 1;
                }
                FillPoly(sides[i],  ET[i][0].y1,  fill_colour[i],  ymax,  i,  init_flag,  poly_status,
polygons);
            }
        }
        for(p = 0; p < MAX_Y; p++)
            for(q = 0; q < MAX_X; q++)
                putpixel(q, revmap_y(p), F[p][q]);
    }
    CoordinateSystem();

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

```
void FillPoly(int sides, int ymin, int fill_colour, int ymax, int poly_nos, int init_flag, status
*poly_status, int polygons)
{
  int y;
  int ET_pointer;
  static char entry = 0;

  active_edge **AET_pointer;
  active_edge *e, *new_edge;

  void ScanFill(active_edge *AET_pointer, int y, int fill_colour, int index);

  if(entry == 0)
  {
    AET_pointer = (active_edge **) malloc(polygons * sizeof(active_edge *));
    entry++;
  }

  if(init_flag == 0)
  {
    y = ymin;
    ET_pointer = 0;
    AET_pointer[poly_nos] = (active_edge *) malloc(sizeof(active_edge));
    AET_pointer[poly_nos]->next = NULL;
  }
  else
  {
    y = poly_status[poly_nos].y;
    ET_pointer = poly_status[poly_nos].ET_pointer;
    AET_pointer[poly_nos] = poly_status[poly_nos].AET_pointer;
  }
  while((ET_pointer != EMPTY || AET_pointer[poly_nos]->next != NULL) && (y <=
ymax))
  {
    DeleteEdge(AET_pointer[poly_nos], y);

    while(ET_pointer != EMPTY && ET[poly_nos][ET_pointer].y1 == y)
    {
        e = AET_pointer[poly_nos];
        while(e->next != NULL)
          e = e->next;
        new_edge = (active_edge*) malloc(sizeof(active_edge));
        new_edge->ymax = ET[poly_nos][ET_pointer].y2;
        new_edge->x = ET[poly_nos][ET_pointer].x1;
        new_edge->dx          =          (float)(ET[poly_nos][ET_pointer].x2          -
ET[poly_nos][ET_pointer].x1)          /          (ET[poly_nos][ET_pointer].y2          -
ET[poly_nos][ET_pointer].y1);
        e->next = new_edge;
        new_edge->next = NULL;
        if(ET_pointer == (sides -1))
          ET_pointer = EMPTY;
```

```
        else
          ET_pointer++;
    }

    if(AET_pointer[poly_nos]->next != NULL)         /* If AET is non empty */
    {
        SortAET(AET_pointer[poly_nos]);
        ScanFill(AET_pointer[poly_nos], y, fill_colour, poly_nos);

        y++;
        UpdateAET(AET_pointer[poly_nos]);
    }
  }
  poly_status[poly_nos].y = y;
  poly_status[poly_nos].ET_pointer = ET_pointer;
  poly_status[poly_nos].AET_pointer = AET_pointer[poly_nos];
}


void ScanFill(active_edge *AET_pointer, int y, int fill_colour, int index)
{
  active_edge *edge = AET_pointer->next;
  active_edge *edge_next = edge->next;

  int x, z;
  char parity = ODD;

  while(edge_next != NULL)
  {
    if(parity == ODD)
    {
        x = ceil(edge->x);
        z = round((- d[index] - (l[index] * x) - (m[index] * y)) / n[index]);
        for(; x <= floor(edge->next->x); x++)
        {
          if( Z[map_y(y)][map_x(x)] >= z )
          {
            Z[map_y(y)][map_x(x)] = z;
            F[map_y(y)][map_x(x)] = fill_colour;
          }
          if(m[index] != 0)
            z = round(z - (l[index]/m[index]));
        }
    }
    edge = edge_next;
    edge_next = edge_next->next;
    parity = !parity;
  }
}
```

```
void swap(int* a, int* b)
{
  int t;
  t = *a;   *a = *b;   *b = t;
}
```

/* Z-Buffer Algorithm with Projection */

```
#include <math.h>

#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\ScanPoly.h"
#include "C:\Programs\Graphics\Common\Project.h"

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))

#define INFINITY 9999
#define MAX_X 640
#define MAX_Y 1
#define LOOP ((getmaxy() + 1) / MAX_Y)

#define PI 3.141592
#define PHI (PI/180) * 45
#define THETA (PI/180) * 54.7

#define map_x(x) (x + getmaxx()/2)
#define map_y(y) (y + getmaxy()/2 - loop_count * MAX_Y)
#define revmap_y(y) (getmaxy() - y - (loop_count * MAX_Y))

int Z[MAX_Y][MAX_X];        /* z-Buffer */
char F[MAX_Y][MAX_X];       /* Frame Buffer */

edge **ET;              /* Edge Table Array */

int loop_count;
float *l, *m, *n, *d;

typedef struct
        {
          int y;
          int ET_pointer;
          active_edge *AET_pointer;
        } status;
```

```
int main()
{
  status *poly_status;
  int ymax, z;
  unsigned int i, j, p, q;
  char *initialised;
  unsigned char *sides, polygons;
  unsigned char *fill_colour, back_colour;
  unsigned char init_flag;
  FILE *fp;

  void FillPoly(int sides, int ymin, int fill_colour, int ymax, int poly_nos, int init_flag, status
*poly_status, int polygons);
  clrscr();

  fp = fopen("C:\\Programs\\Graphics\\Input\\z-Buffer.txt", "rt");
  printf("Set the Background colour : \n");
  ColourMenu();
  fscanf(fp, "%d", &back_colour);

  printf("Enter the nos. of polygons : ");
  fscanf(fp, "%d", &polygons);

  ET = (edge **) malloc(polygons * sizeof(edge *));

  l = (float *) malloc(polygons * sizeof(float));
  m = (float *) malloc(polygons * sizeof(float));
  n = (float *) malloc(polygons * sizeof(float));
  d = (float *) malloc(polygons * sizeof(float));

  sides = (char *) malloc(polygons * sizeof(char));
  fill_colour = (char *) malloc(polygons * sizeof(char));
  poly_status = (status *) malloc(polygons * sizeof(status));

  initialised = (char *) malloc(polygons * sizeof(char));
  for(i = 0; i < polygons;i++)
    initialised[i] = -1;

  if(l == NULL || m == NULL || n == NULL || d == NULL || ET == NULL || sides == NULL
|| initialised == NULL || fill_colour == NULL || poly_status == NULL)
  {
    printf("Memory Allocation unsuccessful.\n");
    getch();
    return EXIT_FAILURE;
  }

  printf("Assume that the polygons are planar with eqn : lx + my +nz + d = 0\n");

  for(i = 0; i< polygons; i++)
  {
```

```
    printf("Enter the coefficients of the equation of the plane of the polygon %d : \n(l, m,
n, d) : ", i + 1);
    fscanf(fp, "%f %f %f %f", &l[i], &m[i], &n[i], &d[i]);

    l[i] = cos(acos(l[i]) + THETA);
    n[i] = cos(acos(n[i]) + PHI);

    printf("Enter the nos. of sides in polygon %d : ", i + 1);
    fscanf(fp, "%d", &sides[i]);

    ET[i] = (edge *) malloc(sides[i] * sizeof(edge));
    if(ET[i] == NULL)
    {
        printf("Memory Allocation unsuccessful.\n");
        getch();
        return EXIT_FAILURE;
    }

    printf("Enter the x and y co-ordinates of polygon %d in sequence.\n", i + 1);
    printf("The z co-ordinate for each vertex will be determined through the equation \nof
the plane : ");
    for(j = 0; j < sides[i]; j++)
    {
        fscanf(fp, "%d %d", &ET[i][j].x1, &ET[i][j].y1);

        z = round((- d[i] - (l[i] * ET[i][j].x1) - (m[i] * ET[i][j].y1)) / n[i]);
        Axonometric(&ET[i][j].x1, &ET[i][j].y1, &z, 1, PHI, THETA);

        ET[i][j-1].x2 =  ET[i][j].x1;
        ET[i][j-1].y2 =  ET[i][j].y1;
    }
    ET[i][j-1].x2 =  ET[i][0].x1;          /* Closing the Polygon */
    ET[i][j-1].y2 =  ET[i][0].y1;

    printf("Enter the colour to be used for filling : \n");
    ColourMenu();
    fscanf(fp, "%d", &fill_colour[i]);

    sides[i] = SetupET(ET[i], sides[i]);
  }

  InitGraph("C:\\tcc\\bgi\\");

  for(loop_count = 1; loop_count <= LOOP; loop_count++)
  {
    for(i = 0; i < MAX_Y; i++)
        for(j = 0; j < MAX_X; j++)
        {
          Z[i][j] = INFINITY;
          F[i][j] = back_colour;
        }
```

```
    ymax = -getmaxy()/2 + (loop_count * MAX_Y);
    for(i = 0; i < polygons; i++)
    {
        if( ET[i][0].y1 <= ymax )
        {
          if(initialised[i] == 1)
            init_flag = 1;
          else
          {
            init_flag = 0;
            initialised[i] = 1;
          }
          FillPoly(sides[i], ET[i][0].y1, fill_colour[i], ymax, i, init_flag, poly_status,
polygons);
        }
    }
    for(p = 0; p < MAX_Y; p++)
        for(q = 0; q < MAX_X; q++)
          putpixel(q, revmap_y(p), F[p][q]);
  }
  CoordinateSystem();

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void FillPoly(int sides, int ymin, int fill_colour, int ymax, int poly_nos, int init_flag, status
*poly_status, int polygons)
{
  int y;
  int ET_pointer;
  static char entry = 0;

  active_edge **AET_pointer;
  active_edge *e, *new_edge;

  void ScanFill(active_edge *AET_pointer, int y, int fill_colour, int index);

  if(entry == 0)
  {
    AET_pointer = (active_edge **) malloc(polygons * sizeof(active_edge *));
    entry++;
  }

  if(init_flag == 0)
  {
    y = ymin;
    ET_pointer = 0;
    AET_pointer[poly_nos] = (active_edge *) malloc(sizeof(active_edge));
```

```
    AET_pointer[poly_nos]->next = NULL;
  }
  else
  {
    y = poly_status[poly_nos].y;
    ET_pointer = poly_status[poly_nos].ET_pointer;
    AET_pointer[poly_nos] = poly_status[poly_nos].AET_pointer;
  }
  while((ET_pointer != EMPTY || AET_pointer[poly_nos]->next != NULL) && (y <=
ymax))
  {
    DeleteEdge(AET_pointer[poly_nos], y);

    while(ET_pointer != EMPTY && ET[poly_nos][ET_pointer].y1 == y)
    {
        e = AET_pointer[poly_nos];
        while(e->next != NULL)
          e = e->next;
        new_edge = (active_edge*) malloc(sizeof(active_edge));
        new_edge->ymax = ET[poly_nos][ET_pointer].y2;
        new_edge->x = ET[poly_nos][ET_pointer].x1;
        new_edge->dx          =          (float)(ET[poly_nos][ET_pointer].x2          -
ET[poly_nos][ET_pointer].x1)          /          (ET[poly_nos][ET_pointer].y2          -
ET[poly_nos][ET_pointer].y1);
        e->next = new_edge;
        new_edge->next = NULL;
        if(ET_pointer == (sides -1))
          ET_pointer = EMPTY;
        else
          ET_pointer++;
    }

    if(AET_pointer[poly_nos]->next != NULL)        /* If AET is non empty */
    {
        SortAET(AET_pointer[poly_nos]);
        ScanFill(AET_pointer[poly_nos], y, fill_colour, poly_nos);

        y++;
        UpdateAET(AET_pointer[poly_nos]);
    }
  }
  poly_status[poly_nos].y = y;
  poly_status[poly_nos].ET_pointer = ET_pointer;
  poly_status[poly_nos].AET_pointer = AET_pointer[poly_nos];
}



void ScanFill(active_edge *AET_pointer, int y, int fill_colour, int index)
{
  active_edge *edge = AET_pointer->next;
```

```
  active_edge *edge_next = edge->next;

  int x, z;
  char parity = ODD;

  while(edge_next != NULL)
  {
    if(parity == ODD)
    {
        x = ceil(edge->x);
        z = round((- d[index] - (l[index] * x) - (m[index] * y)) / n[index]);
        for(; x <= floor(edge->next->x); x++)
        {
          if( Z[map_y(y)][map_x(x)] >= z )
          {
            Z[map_y(y)][map_x(x)] = z;
            F[map_y(y)][map_x(x)] = fill_colour;
          }
          if(m[index] != 0)
            z = round(z - (l[index]/m[index]));
        }
    }
    edge = edge_next;
    edge_next = edge_next->next;
    parity = !parity;
  }
}


void swap(int* a, int* b)
{
  int t;
  t = *a;   *a = *b;   *b = t;
}
```

/* Program for Generation of PC Surface : 16 point form */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\Project.h"

#define PI 3.141592
#define PHI (PI/180) * 45
#define THETA (PI/180) * 54.7


int main()
```

```
{
  char colour;
  int curr_x, curr_y, curr_z;
  float x[4][4], y[4][4], z[4][4];
  float *px, *py, *pz;
  float Fu[1][4], FTw[4][1];
  float u, w, du, dw;
  FILE *fp;

  float * MatMul(float *mat_1, float *mat_2, int row_1, int col_1, int col_2);
  clrscr();

  fp = fopen("C:\\Programs\\Graphics\\Input\\PCSurfce.txt", "rt");
  printf("Enter 16 equi-spaced points on the surface: ( bottom-left to top right ) :\n");
  for(u = 0; u < 4; u++)
    for(w = 0; w < 4; w++)
        fscanf(fp, "%f %f %f", &x[u][w], &y[u][w], &z[u][w]);

  printf("Enter the steps along u and w respectively : ( 0 < du/dw < 1 ) :\n");
  fscanf(fp, "%f %f", &du, &dw);

  printf("Choose the colour of the surface :\n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tcc\bgi\\");
  CoordinateSystem();

  for(u = 0; u <= 1; u += du)
  {
    Fu[0][0] = 1 - (5.5 * u) + (9.0 * u * u) - (4.5 * u * u * u);
    Fu[0][1] = (9.0 * u) - (22.5 * u * u) + (13.5 * u * u * u);
    Fu[0][2] = (-4.5 * u) + (18.0 * u * u) - (13.5 * u * u * u);
    Fu[0][3] = u - (4.5 * u * u) + (4.5 * u * u * u);

    for(w = 0; w <= 1; w += dw)
    {
        FTw[0][0] = 1 - (5.5 * w) + (9.0 * w * w) - (4.5 * w * w * w);
        FTw[1][0] = (9.0 * w) - (22.5 * w * w) + (13.5 * w * w * w);
        FTw[2][0] = (-4.5 * w) + (18.0 * w * w) - (13.5 * w * w * w);
        FTw[3][0] = w - (4.5 * w * w) + (4.5 * w * w * w);


        px = MatMul((MatMul(&Fu[0][0], &x[0][0], 1, 4, 4)), &FTw[0][0], 1, 4, 1);
        curr_x = round(*px);

        py = MatMul((MatMul(&Fu[0][0], &y[0][0], 1, 4, 4)), &FTw[0][0], 1, 4, 1);
        curr_y = round(*py);

        pz = MatMul((MatMul(&Fu[0][0], &z[0][0], 1, 4, 4)), &FTw[0][0], 1, 4, 1);
        curr_z = round(*pz);
```

```
            Axonometric(&curr_x, &curr_y, &curr_z, 1, PHI, THETA);
            PutPixel(curr_x, curr_y, colour);
        }
    }
    getch();
    closegraph();
    return EXIT_SUCCESS;
}


float * MatMul(float *mat_1, float *mat_2, int row_1, int col_1, int col_2)
{
    int i, j, k;
    float sum;
    float *p_product;

    float **product = (float **) malloc(row_1 * sizeof(float *));
    for(i = 0; i < row_1; i++)
        product[i] = (float *) malloc(col_2 * sizeof(float));

    if( product == NULL )
    {
        printf("Memory allocation unsuccessful.\n");
        getch();
        return NULL;
    }

    for(i = 0; i < row_1; i++)
        for(j = 0; j< col_2; j++)
        {
            sum = 0.0;
            for(k = 0; k< col_1; k++)
                sum = sum + ( mat_1[col_1 * i + k] * mat_2[k * col_2 + j] );
            product[i][j] = sum;
        }
    p_product = &product[0][0];
    free(product);
    return(p_product);
}
```

/* Programming for Plotting values of one row matrix vs. another of the same dimension */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"
```

```
int main()
{
  int i, colour;
  int *X, *Y, n;

  void Plot(int n, int* X, int* Y, int colour);

  printf("Enter the nos. of points to be plotted : ");
  scanf("%d", &n);

  X = (int*) malloc(n * sizeof(int));
  Y = (int*) malloc(n * sizeof(int));

  printf("Enter the abscissa values : ");
  for(i=0;i<n;i++)
    scanf("%d", &X[i]);

  printf("Enter the ordiante values : ");
  for(i=0;i<n;i++)
    scanf("%d", &Y[i]);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", colour);

  InitGraph("C:\\tcc\\bgi\\");
  CoordinateSystem();
  Plot(n, X, Y, colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void Plot(int n, int* X, int* Y, int colour)
{
  int i;
  int x1, y1, x2, y2;

  for(i=1;i<n;i++)
  {
    x1 = X[i-1];  y1 = Y[i-1];
    x2 = X[i]  ;  y2 = Y[i]  ;
    BresLine(x1, y1, x2, y2, colour);
  }
}
```

/* Program for Plotting the graph of a function */

```
#include "C:\Programs\Graphics\Common\Graphics.h"
#include "C:\Programs\Graphics\Common\BresLine.h"

#define f(x) (x*x)


int main()
{
  int colour;

  void PlotFunc(int colour);

  printf("Enter the colour to be used for drawing : \n");
  ColourMenu();
  scanf("%d", &colour);

  InitGraph("C:\\tcc\\bgi\\");
  CoordinateSystem();

  PlotFunc(colour);

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void PlotFunc(int colour)
{
  int i, x;
  int *X, *Y;
  int x1, y1, x2, y2;

  X = (int*) malloc(getmaxx() * sizeof(int));
  Y = (int*) malloc(getmaxx() * sizeof(int));

  for(x = -getmaxx()/2, i=0; x<= getmaxx()/2; x++, i++)
  {
    X[i] = x;
    Y[i] = f(x);
  }


  for(i=1;i<= getmaxx();i++)
  {
```

```
    x1 = X[i-1];  y1 = Y[i-1];
    x2 = X[i]  ;  y2 = Y[i]  ;
    if(abs(Y[i]) <= getmaxx()/2 && abs(Y[i-1]) <= getmaxx()/2)
        BresLine(x1, y1, x2, y2, colour);
  }
}
```

/* Program for Generation of Mandelbrot Set */

```
#include <math.h>
#include "C:\Programs\Graphics\Common\Graphics.h"

#define ROWS (getmaxy() + 1)
#define COLUMNS (getmaxx() + 1)

#define ITERATE 500
#define MAX_MOD 2

#define MIN_X -2.5
#define MAX_X 1.5
#define INCR_X (MAX_X - MIN_X) / COLUMNS

#define MIN_Y -1.5
#define MAX_Y ((ROWS * (MAX_X - MIN_X)) / COLUMNS) + MIN_Y      /* For no
distortion */
#define INCR_Y (MAX_Y - MIN_Y) / ROWS

#define SCALE_X COLUMNS / (MAX_X - MIN_X)          /* Map range to actual pixels */
#define SHIFT_X (MIN_X + MAX_X) / (-2 * INCR_X)    /* Shift origin so that reqd. range
comes to center */

#define SCALE_Y ROWS / (MAX_Y - MIN_Y)
#define SHIFT_Y (MIN_Y + MAX_Y) / (-2 * INCR_Y)

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))


typedef struct
        {
          double real;
          double imaginary;
        } complex;


int main()
{
  int iteration_count;
```

```
  complex z, c;

  void Increment(complex *pz, complex c);
  double Modulus(complex z);

  InitGraph("C:\\tcc\\bgi\\");

  for(c.real = MIN_X; c.real <= MAX_X; c.real += INCR_X)
    for(c.imaginary = MIN_Y; c.imaginary <= MAX_Y; c.imaginary += INCR_Y)
    {
        z.real = 0;
        z.imaginary = 0;
        for(iteration_count = 0; iteration_count < ITERATE; iteration_count++)
        {
          Increment(&z, c);
          if((Modulus(z)) > MAX_MOD)
          {
            PutPixel(round(SCALE_X * c.real + SHIFT_X), round(SCALE_Y *
c.imaginary + SHIFT_Y), ((iteration_count % 15) + 1));
            break;
          }
        }
    }

  getch();
  closegraph();
  return EXIT_SUCCESS;
}


void Increment(complex *pz, complex c)
{
  double tmp_real = pz->real;
  pz->real = (pz->real * pz->real) - (pz->imaginary * pz->imaginary) + c.real;
  pz->imaginary = (2 * tmp_real * pz->imaginary) + c.imaginary;
}


double Modulus(complex z)
{
  return(sqrt((z.real * z.real) + (z.imaginary * z.imaginary)));
}
```

/* Program for Generation of Julia Set */

```
#include <math.h>
#include "C:\Programs\Graphics\Common\Graphics.h"
```

```
#define ROWS (getmaxy() + 1)
#define COLUMNS (getmaxx() + 1)

#define ITERATE 500
#define MAX_MOD 2

#define MIN_X -2.0
#define MAX_X 2.0
#define INCR_X (MAX_X - MIN_X) / COLUMNS

#define MIN_Y -1.5
#define MAX_Y ((ROWS * (MAX_X - MIN_X)) / COLUMNS) + MIN_Y      /* For no
distortion */
#define INCR_Y (MAX_Y - MIN_Y) / ROWS

#define SCALE_X COLUMNS / (MAX_X - MIN_X)          /* Map range to actual pixels */
#define SHIFT_X (MIN_X + MAX_X) / (-2 * INCR_X)     /* Shift origin so that reqd. range
comes to center */

#define SCALE_Y ROWS / (MAX_Y - MIN_Y)
#define SHIFT_Y (MIN_Y + MAX_Y) / (-2 * INCR_Y)

#define round(x) (x >= (ceil(x)+ floor(x))/2 ? ceil(x) : floor(x))


typedef struct
        {
          double real;
          double imaginary;
        } complex;


int main()
{
  complex c;

  void JuliaSet(complex c);

  InitGraph("C:\\tcc\\bgi\\");

  c.real = -0.5;       /* A point outside MandelBrot Set */
  c.imaginary = 1.0;

  JuliaSet(c);
  getch();
  cleardevice();

  c.real = 0.3;        /* A point inside MandelBrot Set */
  c.imaginary = 0.5;

  JuliaSet(c);
```

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


void JuliaSet(complex c)
{
  int iteration_count;
  complex tz, z;

  void Increment(complex *pz, complex c);
  double Modulus(complex z);

  for(z.real = MIN_X; z.real <= MAX_X; z.real += INCR_X)
    for(z.imaginary = MIN_Y; z.imaginary <= MAX_Y; z.imaginary += INCR_Y)
    {
        tz.real = z.real;
        tz.imaginary = z.imaginary;
        for(iteration_count = 0; iteration_count < ITERATE; iteration_count++)
        {
          if(Modulus(tz) > MAX_MOD)
          {
            PutPixel(round(SCALE_X * z.real + SHIFT_X), round(SCALE_Y *
z.imaginary + SHIFT_Y), ((iteration_count % 15) + 1));
            break;
          }
          Increment(&tz, c);
        }
    }
}


void Increment(complex *pz, complex c)
{
  double tmp_real = pz->real;
  pz->real = (pz->real * pz->real) - (pz->imaginary * pz->imaginary) + c.real;
  pz->imaginary = (2 * tmp_real * pz->imaginary) + c.imaginary;
}


double Modulus(complex z)
{
  return(sqrt((z.real * z.real) + (z.imaginary * z.imaginary)));
}
```

Input files (assumed to be kept in the directory C:\Programs\Graphics\Input)

1. File ClipPoly.txt

100 0y0 100y-100 0y0 -100n100 -50y50 0y100 50y-100 50y-50 0y-100 -50n * : window - convex, polygon - concave

200 0y0 200y-200 0y0 -200n100 -50y50 0y100 50y-100 50y-50 0y-100 -50n * : window - convex, polygon - concave, fully inside

100 -50y100 50y-100 50y-100 -50n20 -100y20 100y-80 0y-20 -100n * : both convex

200 0y0 200y-200 0y0 -200n75 -200y75 200y0 30y-75 200y-75 -200y0 -30n * : window - convex, polygon - concave

50 -50y50 50y0 50y0 -50n30 -40y30 -20y-20 -20y-20 20y30 20y30 40y-30 40y-30 -40n : Degenerate edge

---

The Sutherland Hodgman Algorithm does not work for concave windows : e.g. :

100 -50y50 0y100 50y-100 50y-50 0y-100 -50n75 -200y75 200y0 30y-75 200y-75 -200y0 -30n : X both - concave

100 -50y50 0y100 50y-100 50y-50 0y-100 -50n20 -100y20 100y-80 0y-20 -100n : X window - concave, polygon - convex

100 100y-100 100y100 -100y100 0y150 0n0 0y150 0y150 150y150 150y0 150n : X window - concave, polygon - convex

---

2. File PCSurface.txt

-75 75 0
-25 75 0
25 75 0
75 75 0
-25 25 0
25 25 0
75 25 0

125 25 0
-75 -25 0
-25 -25 0
25 -25 0
75 -25 0
-25 -75 0
25 -75 0
75 -75 0
125 -75 0
0.002
0.002

****

10 10 10
20 10 20
30 10 10
40 10 20
15 20 30
25 20 20
35 20 20
45 20 50
10 30 40
20 30 30
30 30 10
40 30 30
15 40 30
25 40 20
35 40 10
45 40 40
0.002
0.002

****

-75 75 0
-25 65 0
25 75 0
75 65 0
-25 25 0
25 15 0
75 25 0
125 15 0
-75 -25 0
-25 -35 0
25 -25 0
75 -35 0
-25 -75 0
25 -85 0
75 -75 0
125 -85 0

0.002
0.002

****

80 80 0
75 60 0
70 0 0
75 -60 0
90 -80 0
40 -70 0
0 -75 0
-40 -80 0
-80 -70 0
-85 -60 0
-80 0 0
-75 60 0
-80 80 0
-40 70 0
0 65 0
40 70 0
0.002
0.002

****

10 10 0
20 10 0
30 10 0
40 10 0
15 20 0
25 20 0
35 20 0
45 20 0
10 30 0
20 30 0
30 30 0
40 30 0
15 40 0
25 40 0
35 40 0
45 40 0
0.03
0.03

****

Apart from the 16 points in each set the last 2 parameters supply the step size to be used
to increment along the u and w direction i.e du and dw respectively. Higher values of du and

dw will result in the surface being sparsely populated by points rather than being fully filled.

---

3. File PolyFill.txt

5 30 -30 0 50 -30 -30 40 20 -40 20

7 51 23 69 87 -54 -26 25 -63 25 68 98 75 95 44

3 30 30 50 30 30 90

5 -50 50 -10 20 50 50 50 -30 -30 -50

4 50 -50 -50 50 50 50 -50 -50

6 50 50 -50 50 0 -50 50 -30 0 70 -50 –30

4. File z-Buffer.txt

5 3
0.8 1 1 -3 5 60 -60 0 100 -60 -60 80 40 -80 40 4
0 0.2 1 -2 3 100 -100 0 100 -100 -100 9
1 0 1 0 4 40 40 -40 40 -40 -40 40 -40 6

12 5
0 0 1 2 5 -50 50 -10 20 50 50 50 -30 -30 -50 10
0.8 1 1 -3 5 60 -60 0 100 -60 -60 80 40 -80 40 4
1 1 -1 1 4 50 -50 -50 50 50 50 -50 -50 2
0 0.2 1 -2 3 100 -100 0 100 -100 -100 9
0 1 1 1 7 51 23 69 87 -54 -26 25 -63 25 68 98 75 95 44 7