

NAME: Naman Jain

ENROLLMENT: EN19IT301050

SUBJECT: CGMM

TYPE: Practical File

**INDEX**

**Hardware and Software Requirements**

**Basic Structure of a C-graphics Program**

1. Study libgraph and installation of libgraph and write a small program based on predefined funtions of libgraph.

2. Write a program to implement DDA algorithm and use same to draw a STAR on screen.

3. Write a program to implement a Bresenham line drawing algorithm and using same algorithm to draw a hut on a screen.

4. Write a program to implement Circle using Mid-Point Algorithm and use same algorithm to draw an olympics ring on a screen.

5. Write a program to implement Circle using Bresenham circle drawing algorithm and using same draw solar system on the screen.

6. Write a program to implement the boundary fill algorithm and using the same algorithm to fill color in hut.

7. Write a program to draw a star a on the screen and move that polygon to other part of the screen using the translation transformation.

8. Write a program to draw a boat on the screen and rotate that boat by 90 degree using rotation transformation.

9. Write a program a draw a photo frame on the screen and enlarge that frame by implementing scaling transformation.

10. User has drawn a star and he is interested to see only a half part of it. Write a program to implement that condition using any line clipping algorithm.

11. Project:

Hardware and Software Requirements

For implementing graphics concepts in C-Language:

1. Software Requirement:

Turbo C / C++ compiler that supports graphics.h package.

special DOSBOXed installer for Turbo C++ compiler

Download from following links:

http://www.megaleecher.netDownload\_Turbo\_For\_Windows

2. Minimum hardware requirements:

\* Intel Pentium III800 MHz Processor or higher version

\* Intel chipset 810 mother board or higher version

\* 14’’ color monitor or greater than that

\* Mouse

\* Keyboard

\* 2GB HDD or greater

\* 256 MB RAM or greater

For doing key frame animation:

Software Requirements: Adobe Flash Professional version 6 .

Download Adobe Flash CS6 which contains Flash Professional Also and install

Hardware Requirements:

\* Intel® Pentium® 4, Intel Centrino®, Intel Xeon®, or Intel CoreTM Duo

(or compatible) processor

\* Microsoft® Windows® 7 (64 bit) or Windows 8 (64 bit)

\* 4GB of RAM

\* 2.5GB of available hard-disk space for installation; additional free space required

during installation (cannot install on removable flash storage devices)

\* 1024x768 display (1280x800 recommended)

Basic Structure of a C-graphics Program

#include<stdio.h>

#include<graphics.h>//must be included for every graphics program #include<conio.h>

#include<dos.h> //for including delay function.

void main()

{

int gd=DETECT,gm; //gd=detects best available graphics driver, gm=graphics mode.

initgraph(&gd,&gm,”C:\\TurboC3\\BGI”); // for initializing graph mode

// above 2 steps are must for every graphics program.

//declaration of any variables must be done before calling initgraph() function.

// next write code for producing requiring design or drawing object line(100,100,200,200);//draws a line segment.

getch();

}

Experiment-1

OBJECTIVE: Study libgraph and installation of libgraph and write a small program based on predefined funtions of libgraph.

What is Libgraph?

Libgraph is an implementation of the TurboC graphics API (graphics.h) on GNU/Linux using SDL. In Ubuntu, we need to  install libgraph library that contains **graphics.h** header file and other important files to be able to run any graphics program . **libgraph** is an implementation of the TurboC graphics API (graphics.h) on GNU/Linux using SDL. It is simple, easy-to-use 2D graphics interface - could be used for simple prototyping, visualization or studying graphics algorithms.

**Steps to install libgraph on ubuntu:**

1. After that just download this libgraph installation files from the following link.(Download Libgraph files).

2. Then copy the package to the home directory. And extract it there.

3. Once it extracted open the folder. And right click on empty space and click on ‘Open in terminal’.

4. In terminal execute following command one by one.

. /configure

sudo make install

sudo cp /usr/local/lib/libgraph.\* /usr/lib

5. And done. You have successfully installed libgraph. To check its installation type man libgraph and it will show up a manual page of libgraph.

And one more thing. You can compile a graphics program by typing following.

For C++ program

g++ .cpp - lgraph

For C program

gcc .c - lgraph

SAMPLE PROGRAM

#include<stdio.h>

#include<stdlib.h>

#include<graphics.h>

int main()

{

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

circle(50, 50, 30);

delay(500000);

closegraph();

return 0;

}

**OUTPUT:**



Experiment-2

**OBJECTIVE:** Write a program to implement DDA algorithm and use same to draw a STAR on screen.

DDA (Digital Differential Analyzer) Algorithm

In computer graphics, the **DDA algorithm** is the simplest algorithm among all other line generation algorithms. Here, the **DDA** is an abbreviation that stands for **"Digital Differential Analyzer"**. It is an incremental method, i.e. it works by incrementing the source coordinate points according to the values of the slope generated.

Hence, we can define DDA as follows,

***"DDA stands for Digital Differential Analyzer. This algorithm is incremental and is used for the rasterization of lines, triangles, and polygons."***

### **Working of the DDA Algorithm**

Suppose we have to **draw a line PQ with coordinates P (x1, y1) and Q (x2, y2)**.

1. First, Calculate **dx = (x2 - x1) and dy = (y2 - y1)**
2. Now calculate the slope **m = (dy / dx)**
3. Calculate the number of points to be plotted (i.e. **n**) by finding the maximum of **dx** and **dy**, i.e. **n = abs (max (dx , dy))**  
   To draw an accurate line, more number of points are required. Therefore, the maximum of the two values is used here.
4. Now as the n is calculated, to know by how much each source point should be incremented, calculate **xinc** and **yinc** as follows: **xinc = (dx / n) and yinc = (dy / n)**
5. Now we draw the points from **P** to **Q**. The successive points are calculated as follows: **(xnext, ynext) = (x + xinc, y + yinc)**  
   Start plotting the points from **P** and stop when **Q** is reached. In case the incremented values are decimal, use the round off values.

ALGORITHM:

1. Start.

2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare gdriver=DETECT, mode.

3. Initialize the graphic mode with the path location in TurboC3 folder.

4. Input the two line end-points and store the left end-points in (x1,y1).

5. Load (x1, y1) into the frame buffer; that is, plot the first point. put x=x1,y=y1. 6. Calculate dx=x2-x1 and dy=y2-y1.

7. If abs (dx) > abs (dy), do s=abs(dx).

8. Otherwise s= abs(dy).

9. Then xi=dx/s and yi=dy/s.

10. Start from k=0 and continuing till k<s,the points will be

i. x=x+xi.

ii. Y=y+yi.

11. Plot pixels using putpixel at points (x,y) in specified colour.

12. Close Graph and stop.

### **Advantages of the DDA algorithm**

Now, we will be looking at the advantages that the DDA algorithm offers over other line drawing algorithms.

1. It is the simplest line generation algorithm.
2. Implementation of the DDA algorithm is very easy as compared to other line generation algorithms.
3. It does not use multiplication which reduces the time complexity of implementation.
4. It is a faster and a better method than using the direct method of the line equation: i.e. **y = mx + c**

### **Disadvantages of the DDA algorithm**

* DDA algorithm use floating-point arithmetic as it involves the use of division in the calculation of xinc and yinc. This floating-point arithmetic makes the algorithm time-consuming.
* The use of floating-point arithmetic decreases the accuracy of the generated points. Hence the points that we get are not accurate, i.e. they may not lie accurately on the line.
* As the points that we get from the DDA algorithm are not accurate, the lines generated by this algorithm are not smooth, i.e. some discontinuation and zigzag nature can be commonly seen in the lines drawn through this algorithm.

**PROGRAM 2.1**

#include<graphics.h>

#include<iostream.h>

#include<math.h>

#include<dos.h>

#include<conio.h>

void main( )

{

float x,y,x1,y1,x2,y2,dx,dy,step;

int i,gd=DETECT,gm;

initgraph(&gd,&gm,”C:\\TURBOC3\\BGI”);

cout<<"Enter the value of x1 and y1 : ";

cin>>x1>>y1;

cout<<"Enter the value of x2 and y2: ";

cin>>x2>>y2;

dx=abs(x2-x1);

dy=abs(y2-y1);

if(dx>=dy)

step=dx;

else

step=dy;

dx=dx/step;

dy=dy/step;

x=x1;

y=y1;

i=1;

while(i<=step)

{

putpixel(x,y,5);

x=x+dx;

y=y+dy;

i=i+1;

delay(100);

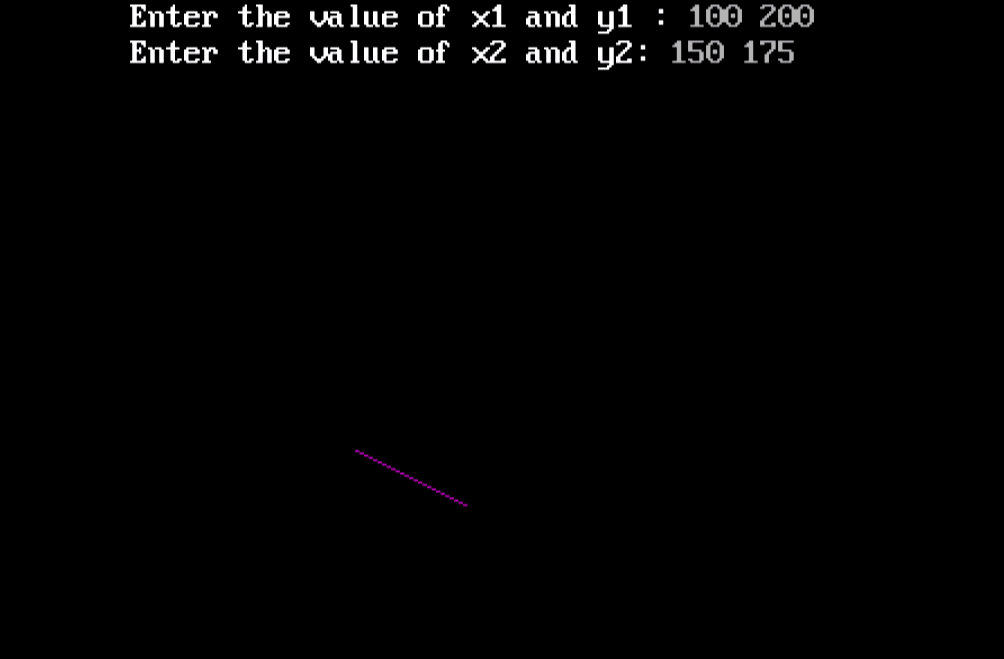
}

getch();

closegraph();

}

**OUTPUT 2.1:**



**PROGRAM 2.2**

#include<iostream>

#include<graphics.h>

using namespace std;

void star(float,float,float,float);

int main()

{

    int i,gd=DETECT,gm;

    char path[] = "";

    initgraph(&gd,”C:\\TURBOC3\\BGI”);

    star(300,200,200,400);

    star(200,400,400,400);

    star(300,200,400,400);

    star(200,250,400,250);

    star(200,250,300,450);

    star(400,250,300,450);

    getch();

    closegraph();

}

void star(float x1,float y1,float x2,float y2)

{

    float x,y,dx,dy,step;

    int i;

    dx=(x2-x1);

    dy=(y2-y1);

    if(dx>=dy)

    {

        step=abs(dx);

    }

    else

    {

       step=abs(dy);

    }

    dx=dx/step;

    dy=dy/step;

    x=x1;

    y=y1;

    i=1;

    while(i<=step)

    {

        putpixel(x,y,WHITE);

        x=x+dx;

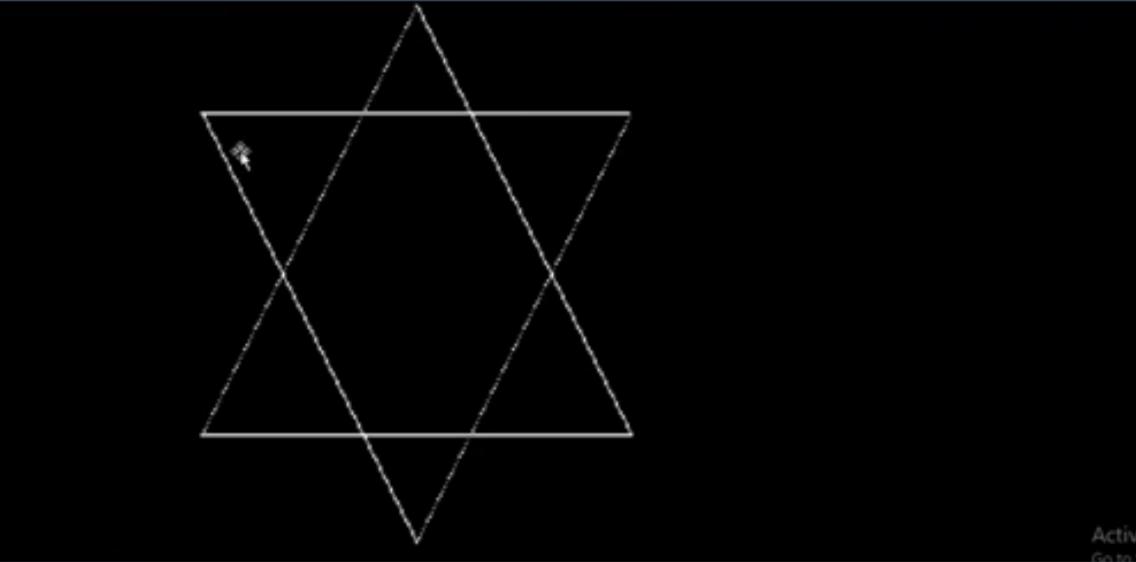
        y=y+dy;

        i=i+1;

    }

}

**OUTPUT 2.2**



Experiment-3

**OBJECTIVE**: Write a program to implement a Bresenham line drawing algorithm and using same algorithm to draw a hut on a screen.

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.

**Advantages of Bresenham Line Drawing Algorithm-**

The advantages of Bresenham Line Drawing Algorithm are-

* It is easy to implement.
* It is fast and incremental.
* It executes fast but less faster than DDA Algorithm.
* The points generated by this algorithm are more accurate than DDA Algorithm.
* It uses fixed points only.

**Disadvantages of Bresenham Line Drawing Algorithm-**

The disadvantages of Bresenham Line Drawing Algorithm are-

* Though it improves the accuracy of generated points but still the resulted line is not smooth.
* This algorithm is for the basic line drawing.
* It can not handle diminishing jaggies.

**Algorithm of Bresenham’s Line Drawing Algorithm**

**Step 1:**Start.

**Step 2:**Now, we consider Starting point as **(x1, y1)**and ending point**(x2, y2).**

**Step 3:**Now, we have to calculate **?x**and **?y.**

**?x**= **x2-x1**

**?y**= **y2-y1**

**m = ?y/?x**

**Step 4:**Now, we will calculate the decision parameter **pk**with following formula.

 **pk** = **2?y-?x**

**Step 5:**The initial coordinates of the line are**(xk, yk),**and the next coordinates are **(xk+1, yk+1).**Now, we are going to calculate two cases for decision parameter**pk**

**Case 1:**If

**pk< 0**

              Then

**pk+1=pk+2?y**

**xk+1= xk+1**

**yk+1= yk**

**Case 2:**If

**pk>= 0**

              Then

**pk+1=pk+2?y-2?x**

**xk+1=xk+1**

**yk+1=yk +1**

**Step 6:**We will repeat step 5 until we found the ending point of the line and the total number of iterations =**?x-1.**

**Step 7:**Stop.

**PROGRAM 3.1**

#include <iostream.h>

#include <conio.h>

#include <graphics.h>

#include<dos.h>

void bsline(int x,int y,int x2,int y2)

{

int dx,dy,p;

dx=x2-x;

dy=y2-y;

p = 2 \* (dy) - (dx);

while(x <= x2)

{

if(p < 0)

{

x=x+1;

y=y;

p = p + 2 \* (dy);

}

else

{

x=x+1;

y=y+1;

p = p + 2 \* (dy - dx);

}

putpixel(x,y,RED);

delay(10);

}

}

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

int x1,x2,y1,y2;

cout<<"Enter the x1,y1,x2,y2 values : ";

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

bsline(x1,y1,x2,y2);

getch();

closegraph();

}

**OUTPUT 3.1**

****

**PROGRAM 3.2**

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

void main(){

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

printf("\t\t \*\*\* HUT \*\*\*\*");

line(150,100,50,200);

line(150,100,350,100);

line(150,100,300,200);

line(300,200,500,200);

line(350,100,500,200);

line(50,200,300,200);

rectangle(50,400,300,200);

rectangle(300,200,500,400);

rectangle(130,250,230,400);

getch();

closegraph();

}

**OUTPUT 3.2**

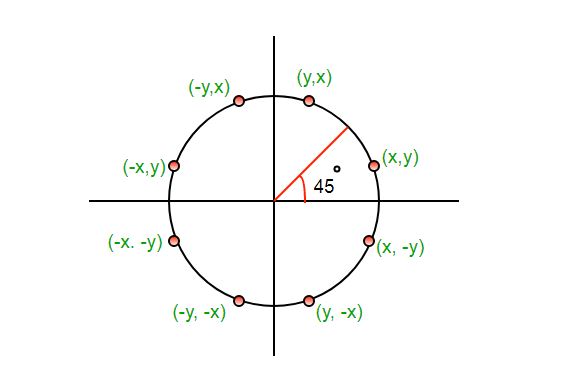


**Experiment-4**

**OBJECTIVE:** Write a program to implement Circle using Mid-Point Algorithm and use same algorithm to draw an olympics ring on a screen.

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about it’s centre.



The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.

**The Mid-Point Circle Drawing Algorithm**

Step 1: Start.

Step 2: Declare x, y, r, xc, yc, P as variables, where (xc, yc) are coordinates of the center.

Step 3: Put x = 0 and y = r

Step 4: Repeat the steps while x ≤ y;

Step 5: Plot (x, y).

Step 6: if (P < 0):

                 Set P = P + 2x + 3

             else if (P >= 0):

                  Set P = P + 2(x-y) + 5

                   y = y - 1

 Step 7: Do x = x + 1

Step 8: End

**PROGRAM 4.1**

#include<iostream.h>

#include<graphics.h>

void drawcircle(int x0, int y0, int radius)

{

    int x = radius;

    int y = 0;

    int err = 0;

    while (x >= y)

    {

putpixel(x0 + x, y0 + y, 7);

putpixel(x0 + y, y0 + x, 7);

putpixel(x0 - y, y0 + x, 7);

putpixel(x0 - x, y0 + y, 7);

putpixel(x0 - x, y0 - y, 7);

putpixel(x0 - y, y0 - x, 7);

putpixel(x0 + y, y0 - x, 7);

putpixel(x0 + x, y0 - y, 7);

if (err <= 0)

{

    y += 1;

    err += 2\*y + 1;

}

if (err > 0)

{

    x -= 1;

    err -= 2\*x + 1;

}

    }

}

int main()

{

int gdriver=DETECT, gmode, error, x, y, r;

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

cout<<"Enter radius of circle: ";

cin>>r;

cout<<"Enter co-ordinates of center(x and y): ";

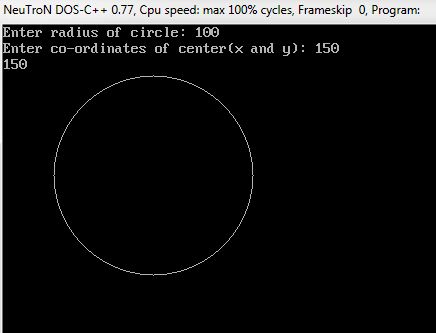
cin>>x>>y;

drawcircle(x, y, r);

return 0;

}

**OUTPUT 4.1**



**PROGRAM 4.2**

#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<math.h>

#include<dos.h>

void main()

{

int i=0,j=0,k=0,l=0,m=0,ch;

float pi=3.1424,a,b,c,d,e;

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

printf("\n\nEnter 1 or 2 ");

scanf("%d",&ch);

printf("\n\nYou have entered %d",ch);

getch();

clrscr();

switch(ch)

{

case 1 : while(i<360)

{

a=(pi/180)\*i;

setcolor(3);

circle(120+100\*sin(a),150-100\*cos(a),10);

i++;

delay(5);

}

while(j<360)

{

b=(pi/180)\*j;

setcolor(0);

circle(280+100\*sin(b),150-100\*cos(b),10);

j++;

delay(5);

}

while(k<360)

{

c=(pi/180)\*k;

setcolor(4);

circle(440+100\*sin(c),150-100\*cos(c),10);

k++;

delay(5);

}

while(l<360)

{

d=(pi/180)\*l;

setcolor(14);

circle(200+100\*sin(d),300-100\*cos(d),10);

l++;

delay(5);

}

while(m<360)

{

e=(pi/180)\*m;

setcolor(2);

circle(370+100\*sin(e),300-100\*cos(e),10);

m++;

delay(5);

}

break;

case 2 : while(i<360)

{

a=(pi/180)\*i;

setcolor(3);

circle(120+100\*sin(a),150-100\*cos(a),10);

i++;

delay(5);

}

while(l<360)

{

d=(pi/180)\*l;

setcolor(14);

circle(200+100\*sin(d),300-100\*cos(d),10);

l++;

delay(5);

}

while(j<360)

{

b=(pi/180)\*j;

setcolor(0);

circle(280+100\*sin(b),150-100\*cos(b),10);

j++;

delay(5);

}

while(k<360)

{

c=(pi/180)\*k;

setcolor(4);

circle(440+100\*sin(c),150-100\*cos(c),10);

k++;

delay(5);

}

while(m<360)

{

e=(pi/180)\*m;

setcolor(2);

circle(370+100\*sin(e),300-100\*cos(e),10);

m++;

delay(5);

}

break;

default:

setcolor(13);

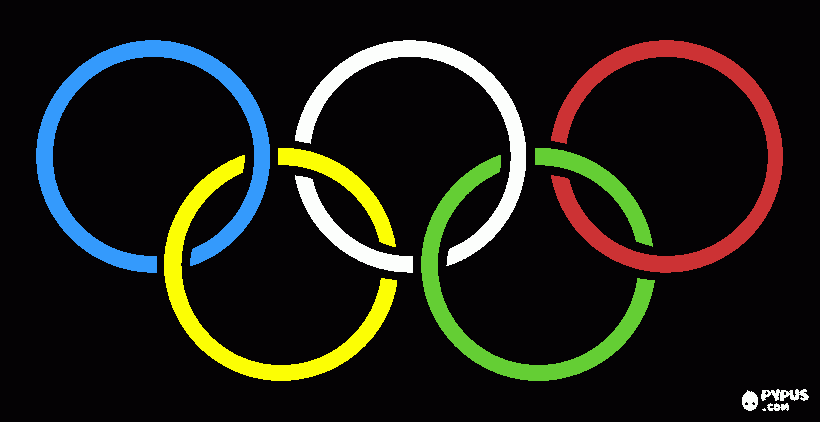
outtextxy(190,220,"YOU HAVE ENTERED THE WRONG CHOICE!!");

}

getch();

}

**OUTPUT 4.2**

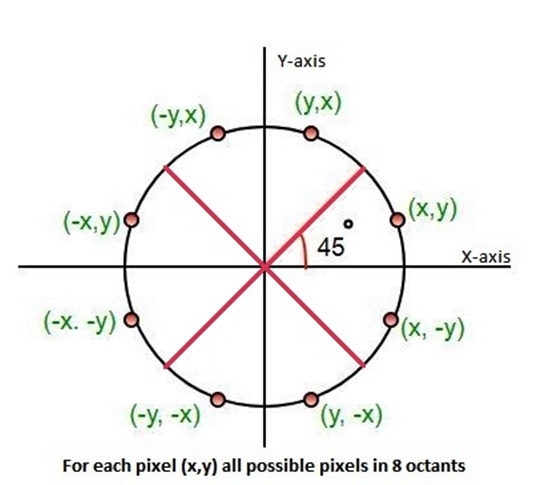


**Experiment-5**

**OBJECTIVE:** Write a program to implement Circle using Bresenham circle drawing algorithm and using same draw solar system on the screen.

The **Bresenham's circle drawing algorithm** is a circle drawing algorithm which calculates all the nearest points nearest to the circle boundary. It is an incremental method (i.e. we increment one of the coordinates of the point and calculate the other coordinate according to it. In this manner we find all the points of that particular polygon). It only uses integer arithmetic which makes it's working faster as well as less complex. The strategy that is followed in this algorithm is to select the pixel which has the least distance with the true circle boundary and with then keep calculating the successive points on the circle.

As we know that the circle follows 8 symmetry property, i.e. if we know the boundary coordinates of the first octant, the rest 7 octant’s value can be easily calculated by changing their magnitudes or by interchanging the coordinate values according to the respective octants. This can be well illustrated by the following diagram,



**Bresenham's Circle Drawing Algorithm**

Step 1: Start.

Step 2: Declare x, y, r, xc, yc and d as variables, where ( xc, yc) are coordinates of the center.

Step 3: Calculate the decision parameter as follows: D = 3 - 2r

Step 4: Initialize       x = 0 , y = r

Step 5:  If x >= y

                Plot eight points by using concepts of eight-way symmetry.

Step 6:  If D < 0  
            then D = D + 4x + 6  
                     x = x + 1  
            If D ≥ 0  
            then D = D+ 4 (x - y) + 10  
                      x = x + 1  
                      y = y - 1

Step 7: End.

**Advantages of Bresenham's Circle Drawing Algorithm**

1. The Bresenhem’s circle drawing algorithm uses integer arithmetic which makes the implementation less complex.
2. Due to its integer arithmetic, it is less time-consuming.
3. This algorithm is more accurate than any other circle drawing algorithm as it avoids the use of round off function.

**Disadvantages of Bresenham's Circle Drawing Algorithm**

1. This algorithm does not produce smooth results due to its integer arithmetic as it fails to diminish the zigzags completely.
2. The Bresenhem’s circle drawing algorithm is not accurate in the case of drawing of complex graphical images.

**PROGRAM 5.1**

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

    void  EightWaySymmetricPlot(int xc,int yc,int x,int y)

   {

    putpixel(x+xc,y+yc,RED);

    putpixel(x+xc,-y+yc,YELLOW);

    putpixel(-x+xc,-y+yc,GREEN);

    putpixel(-x+xc,y+yc,YELLOW);

    putpixel(y+xc,x+yc,12);

    putpixel(y+xc,-x+yc,14);

    putpixel(-y+xc,-x+yc,15);

    putpixel(-y+xc,x+yc,6);

   }

    void BresenhamCircle(int xc,int yc,int r)

   {

    int x=0,y=r,d=3-(2\*r);

    EightWaySymmetricPlot(xc,yc,x,y);

    while(x<=y)

     {

      if(d<=0)

             {

        d=d+(4\*x)+6;

      }

     else

      {

        d=d+(4\*x)-(4\*y)+10;

        y=y-1;

      }

       x=x+1;

       EightWaySymmetricPlot(xc,yc,x,y);

      }

    }

    int  main(void)

   {

    int xc,yc,r,gdriver = DETECT, gmode, errorcode;

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

     errorcode = graphresult();

      if (errorcode != grOk)

     {

        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");

        getch();

        exit(1);

     }

       printf("Enter the values of xc and yc :");

       scanf("%d%d",&xc,&yc);

       printf("Enter the value of radius  :");

       scanf("%d",&r);

       BresenhamCircle(xc,yc,r);

     getch();

     closegraph();

     return 0;

    }

**OUTPUT 5.1**

****

**PROGRAM 5.2**

#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<math.h>

#include<dos.h>

int main()

{

int m=0,

v=260,

eh=300,

mr=100,

j=10,

s=230,

u=190,

n=20;

float pi=3.1424,a,b,c,d,e,f,g,h,z;

int gd=DETECT,gm;

initgraph(&gd,&gm,"C://TurboC3//BGI");

while(1)

{

a=(pi/180)\*m++;

b=(pi/180)\*v++;

c=(pi/180)\*eh++;

d=(pi/180)\*mr++;

e=(pi/180)\*j++;

f=(pi/180)\*s++;

g=(pi/180)\*u++;

h=(pi/180)\*n++;

n++;

cleardevice();

circle(320,240,20);

circle(320+60\*sin(a),240-35\*cos(a),8);

ellipse(320,240,0,360,60,35);

circle(320+100\*sin(b),240-60\*cos(b),12);

ellipse(320,240,0,360,100,60);

circle(320+130\*sin(c),240-80\*cos(c),10);

ellipse(320,240,0,360,130,80);

circle(320+170\*sin(d),240-100\*cos(d),11);

ellipse(320,240,0,360,170,100);

circle(320+200\*sin(e),240-130\*cos(e),14);

ellipse(320,240,0,360,200,130);

circle(320+230\*sin(f),240-155\*cos(f),12);

ellipse(320,240,0,360,230,155);

circle(320+260\*sin(g),240-180\*cos(g),9);

ellipse(320,240,0,360,260,180);

circle(320+280\*sin(h),240-200\*cos(h),9);

ellipse(320,240,0,360,280,200);

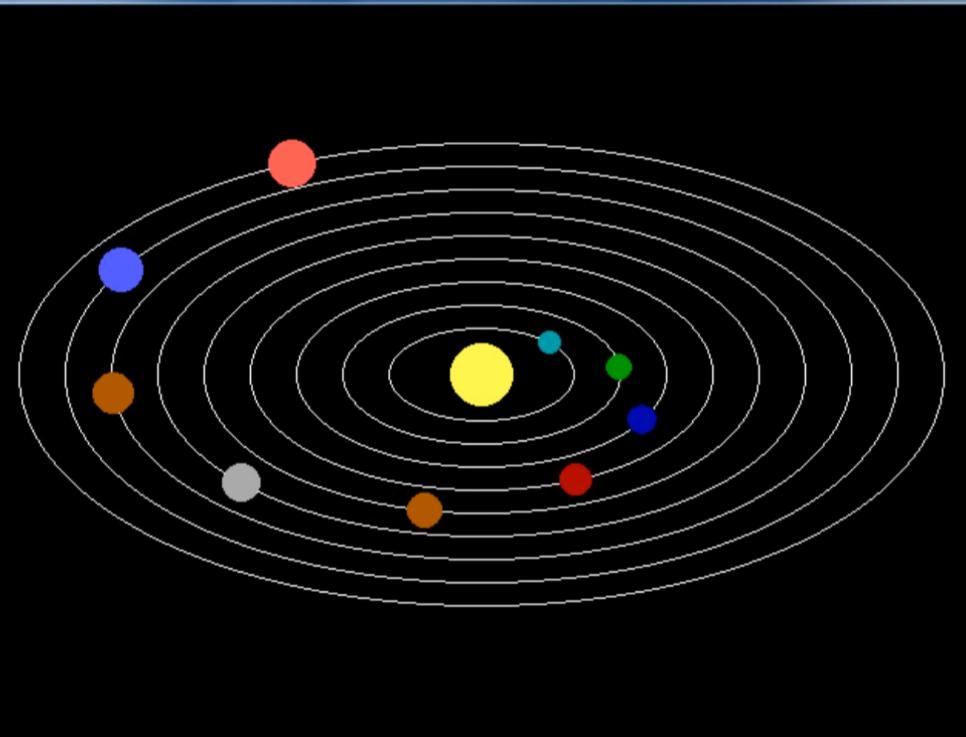
delay(50);

}

getch();

}

**OUTPUT 5.2**



**Experiment-6**

**OBJECTIVE:** Write a program to implement the boundary fill algorithm and using the same algorithm to fill color in hut.

This is an area filling algorithm. This is used where we have to do an interactive painting in computer graphics, where interior points are easily selected. If we have a specified boundary in a single color, then the fill algorithm proceeds pixel by pixel until the boundary color is encountered. This method is called the **boundary-fill algorithm**.

**In this, generally two methods are given that are:**

1. **4-connected:**  
   In this firstly there is a selection of the interior pixel which is inside the boundary then in reference to that pixel, the adjacent pixel will be filled up that is top-bottom and left-right.
2. **8-connected:**  
   This is the best way of filling the color correctly in the interior of the area defined. This is used to fill in more complex figures. In this four diagonal pixel are also included with a reference interior pixel (including top-bottom and left right pixels).

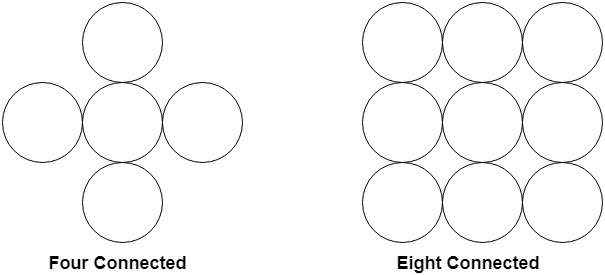
ALGORITHM:

The following steps illustrate the idea of the recursive boundaryfill algorithm:

1. Start from an interior point.

2. If the current pixel is not already filled and if it is not an edge point, then set the pixel with the fill color, and store its neighboring pixels (4 or 8-connected). Store only neighboring pixel that is not already filled and is not an edge point.

3. Select the next pixel from the stack, and continue with step 2.



In 4 connected approach, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.

In 8 connected approach, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel.

**PROGRAM 6.1**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void boundary\_fill(int x, int y, int fcolor, int bcolor) {

if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))

{

delay(10);

putpixel(x, y, fcolor);

boundary\_fill(x + 1, y, fcolor, bcolor);

boundary\_fill(x - 1, y, fcolor, bcolor);

boundary\_fill(x, y + 1, fcolor, bcolor);

boundary\_fill(x, y - 1, fcolor, bcolor);

} }

void main() {

int x, y, fcolor, bcolor;

int gd=DETECT,gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

printf("Enter the seed point (x,y) : ");

scanf("%d%d", &x, &y);

printf("Enter boundary color : ");

scanf("%d", &bcolor);

printf("Enter new color : ");

scanf("%d", &fcolor);

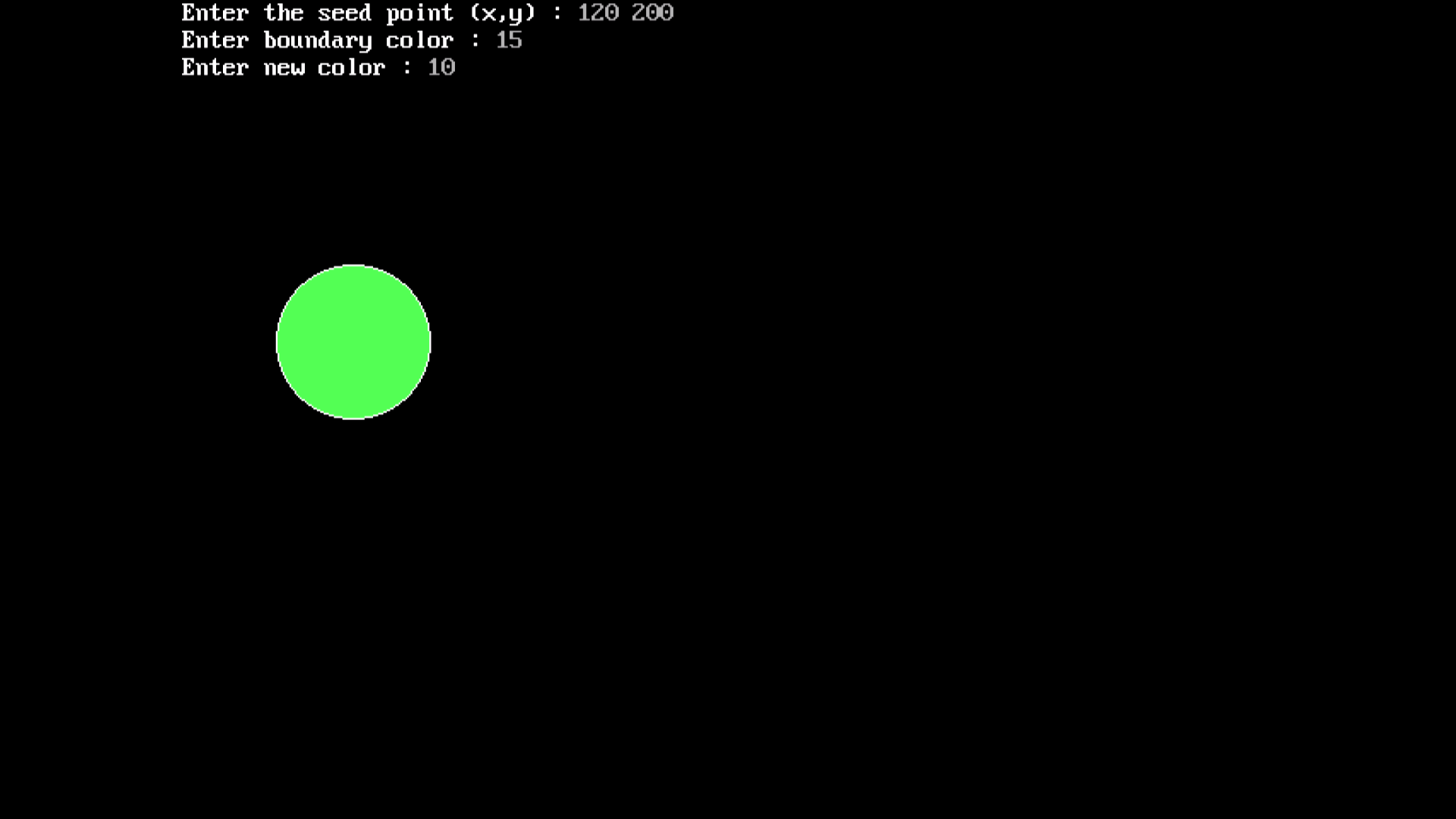
circle(100,200,45);

boundary\_fill(x,y,fcolor,bcolor);

getch();

}

**OUTPUT 6.1**

****

**PROGRAM 6.2**

#include <graphics.h>

#include<conio.h>

void boundaryFill8(int x, int y, int fill\_color,int boundary\_color)

{

if(getpixel(x, y) != boundary\_color &&

getpixel(x, y) != fill\_color)

{

putpixel(x, y, fill\_color);

boundaryFill8(x + 1, y, fill\_color, boundary\_color);

boundaryFill8(x, y + 1, fill\_color, boundary\_color);

boundaryFill8(x - 1, y, fill\_color, boundary\_color);

boundaryFill8(x, y - 1, fill\_color, boundary\_color);

boundaryFill8(x - 1, y - 1, fill\_color, boundary\_color);

boundaryFill8(x - 1, y + 1, fill\_color, boundary\_color);

boundaryFill8(x + 1, y - 1, fill\_color, boundary\_color);

boundaryFill8(x + 1, y + 1, fill\_color, boundary\_color);

}

}

int main()

{

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

setcolor(WHITE);

rectangle(150,180,250,300);

rectangle(250,180,420,300);

rectangle(180,250,220,300);

line(200,100,150,180);

line(200,100,250,180);

line(200,100,370,100);

line(370,100,420,180);

setfillstyle(SOLID\_FILL, BROWN);

boundaryFill8(155, 185, 6, 15);

boundaryFill8(255, 185, 4, 15);

setfillstyle(SLASH\_FILL, BLUE);

boundaryFill8(185, 255, 8, 15);

setfillstyle(HATCH\_FILL, GREEN);

boundaryFill8(200, 105, 4, 15);

boundaryFill8(210, 105, 4, 15);

getch();

closegraph();

return 0;

}

**OUTPUT 6.2**



**Experiment-7**

**OBJECTIVE**: Write a program to draw a star a on the screen and move that polygon to other part of the screen using the translation transformation.

❖ Translation: Translation is defined as moving the object from

one position to another position along straight line path.

❖ We can move the objects based on translation distances along x

and y axis. tx denotes translation distance along x-axis and ty

denotes translation distance along y axis.

❖ Translation Distance: It is nothing but by how much units we

should shift the object from one location to another along x, yaxis.

❖ Consider (x,y) are old coordinates of a point. Then the new

coordinates of that same point (x’,y’) can be obtained as

follows:

X’=x+tx

Y’=y+ty

We denote translation transformation as P. we express above

equations in matrix form as

x,y---old coordinates

x’,y’—new coordinates after

translation

tx,ty—translation distances, T is

**PROGRAM 7**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>

#include<math.h>

void main()

{

int gd = DETECT, gm;

int tx, ty;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI ");

setcolor(5);

line(150,100,100,200);

line(100,200,200,200);

line(200,200,150,100);

line(100,125,200,125);

line(100,125,150,225);

line(150,225,200,125);

printf("Enter translation distance tx, ty:\n");

scanf("%d%d",&tx,&ty);

setcolor(7);

line(tx+150,ty+100,tx+100,ty+200);

line(tx+100,ty+200,tx+200,ty+200);

line(tx+200,ty+200,tx+150,ty+100);

line(tx+100,ty+125,tx+200,ty+125);

line(tx+100,ty+125,tx+150,ty+225);

line(tx+150,ty+225,tx+200,ty+125);

getch();

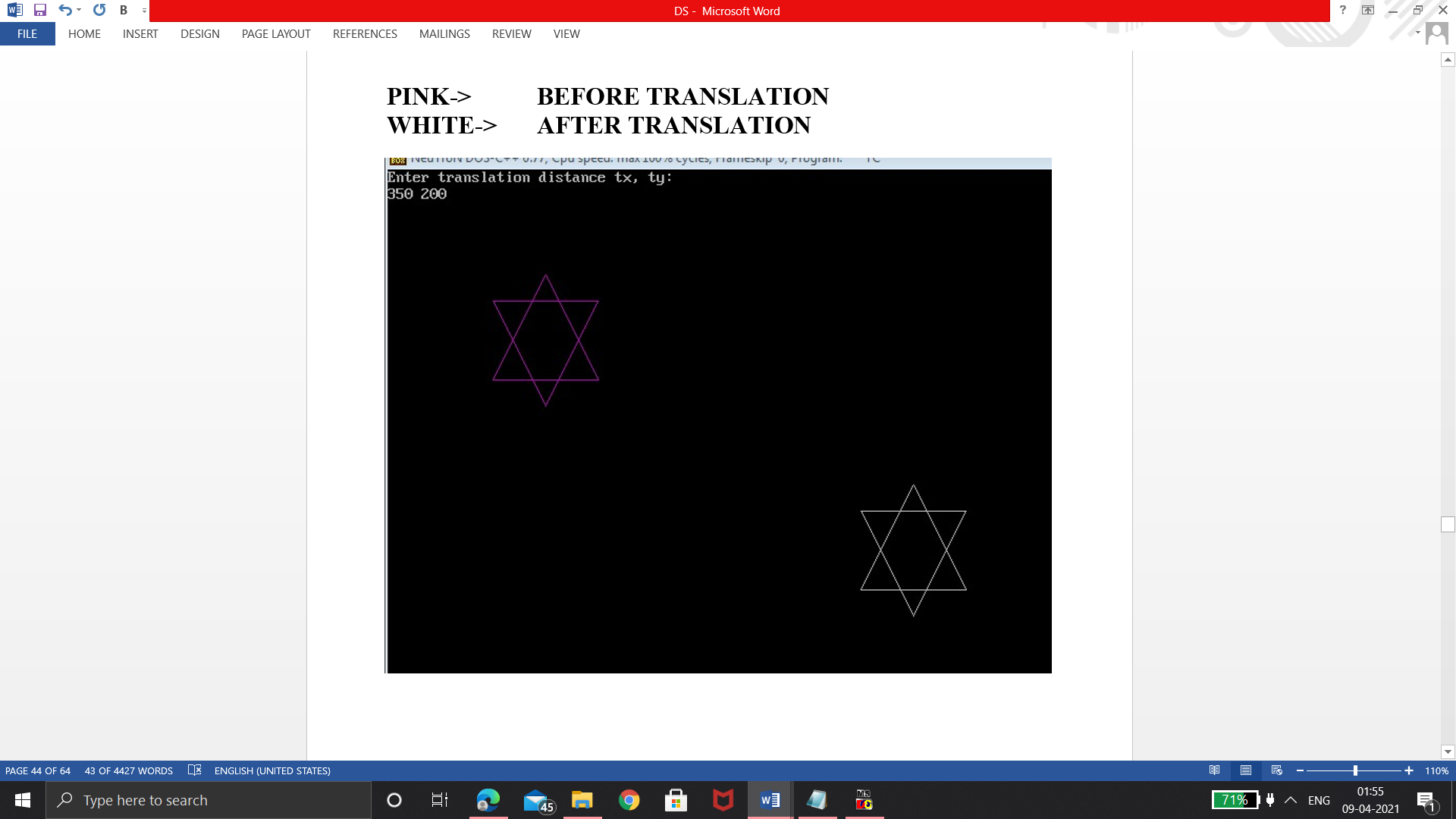
cleardevice();

}

**OUTPUT 7**

**PINK-> BEFORE TRANSLATION**

**WHITE-> AFTER TRANSLATION**



**Experiment-8**

**OBJECTIVE**: Write a program to draw a boat on the screen and rotate that boat by 90 degree using rotation transformation.

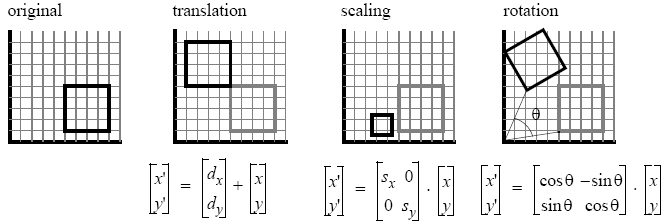
**Rotation** is a type of transformation that is very often used in computer graphics and image processing. Rotation is a process of rotating an object concerning an angle in a two-dimensional plane.

It is a process of changing the angle of the object which can be clockwise or anticlockwise, while we have to specify the angle of rotation and rotation point. A rotation point is also called a pivot point.

There are two types of rotations according to the direction of the movement of the object. These are:

* Anti-clockwise rotation
* Clockwise rotation

The positive value of the rotation angle rotates an object in an anti-clockwise direction while the negative value of the rotation angle rotates an object in a clockwise direction. When we rotate any object, then every point of that object is rotated by the same angle. For example, a straight line is rotated by the endpoints with the same angle and the line is re-drawn between the new endpoints. Also, the polygon is rotated by shifting every vertex with the help of the same rotational angle. Same for circle also, it can be obtained by center position by the specified angle.

v

**PROGRAM 8**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>

#include<math.h>

void main()

{

int gd = DETECT, gm;

int a,t;

initgraph(&gd, &gm, “C:\\TURBOC3\\BGI ");

setlinestyle(SOLID\_LINE, 1, 2);

setcolor(BROWN);

setfillstyle(SOLID\_FILL, BROWN);

sector(50, 100, 180, 360, 50, 15);

setcolor(DARKGRAY);

setlinestyle(SOLID\_LINE, 1, 3);

printf("enter the angle of rotation\n");

scanf("%d",&a);

t=a\*(3.14/180);

setlinestyle(SOLID\_LINE, 1, 2);

setcolor(GREEN);

setfillstyle(SOLID\_FILL, GREEN);

sector(50, 100, 180+t, 360+t, 50, 15);

setcolor(DARKGRAY);

setlinestyle(SOLID\_LINE, 1, 3);

getch();

cleardevice();

}

**OUTPUT 8**

****

****

**Experiment-9**

**OBJECTIVE**: Write a program a draw a photo frame on the screen and enlarge that frame by implementing scaling transformation.

A scaling transformation changes the size of an object.  
Suppose we want the point (x1 y1) to be scaled by a factor sx and by a factor sy along y  
direction.  
Then the new coordinates become : x2 = x1 \* sx and y2 = y1 \* sy

**Algorithm for scaling transformation:**

1. Enter the coordinates of object.  
2. Enter the scaling factor x sx, and for y axis , sy  
3. Multiply the scaling factor sx and sy with the polygon coordinates x1, y1 and getting a new  
coordinates.  
4. Draw an original object.  
5. Draw a scaled object.  
6.Exit

See the source image

**PROGRAM 9**

#include<bits/stdc++.h>

#include<graphics.h>

#include<conio.h>

using namespace std;

int main(){

int gd = DETECT, gm;

initgraph(&gd, &gm, “C:\\TURBOC3\\BGI”);

int mx = getwindowwidth();

int my = getwindowheight();

int left = mx / 2 - 50;

int top = my / 2 - 50;

int right = mx / 2 + 50;

int bottom = my / 2 + 50;

rectangle(left, top, right, bottom);

cout <<"Enter the scaling factors Sx and Sy ";

float sx, sy;

cin >> sx >> sy;

left = left \* sx;

right = right \* sx;

top = top \* sy;

bottom = bottom \* sy;

cleardevice();

rectangle(left, top, right, bottom);

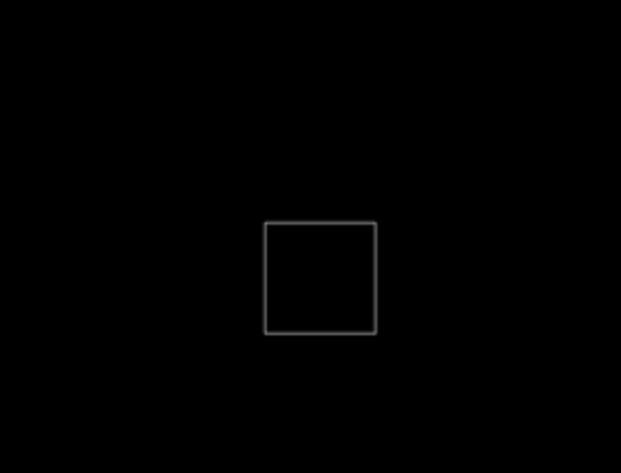
getch();

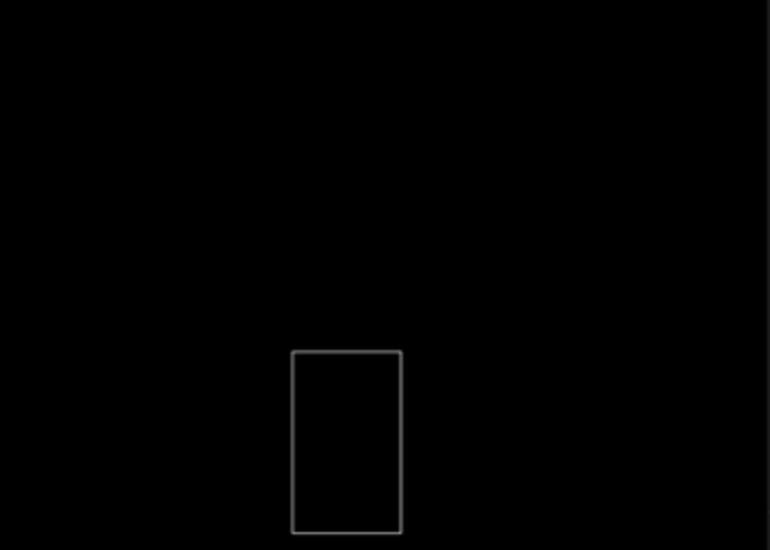
closegraph();

return 0;

}

**OUTPUT 9**





**Experiment-10**

**OBJECTIVE:** User has drawn a star and he is interested to see only a half part of it. Write a program to implement that condition using any line clipping algorithm.

# **Line Clipping:**

It is performed by using the line clipping algorithm. The line clipping algorithms are:

1. Cohen Sutherland Line Clipping Algorithm
2. Midpoint Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

**Cohen Sutherland Line Clipping Algorithm:**

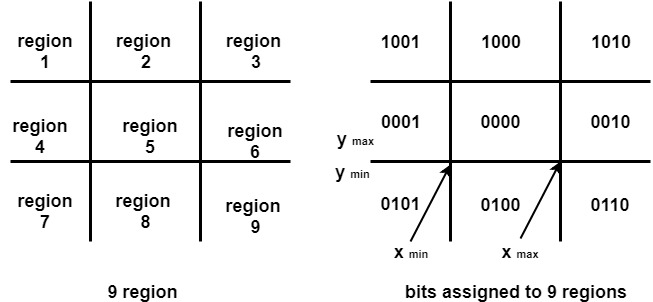
In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

1. **Visible:** If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

**2. Not Visible:** If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x1,y2) and B (x2,y2) are endpoints of line.

**3. Clipping Case:** If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



**Advantage of Cohen Sutherland Line Clipping:**

1. It calculates end-points very quickly and rejects and accepts lines quickly.
2. It can clip pictures much large than screen size.

See the source image

**PROGRAM 10**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void dda(int,int,int,int,int,int,int,int);

void dda(int x1,int y1,int x2,int y2,int xmin,int ymin,int xmax,int ymax)

{

float dx,dy;

float steps,x=x1,y=y1;

int i=0;

dx=x2-x1;

dy=y2-y1;

if(abs(dx)>=abs(dy))

steps=abs(dx);

else

steps=abs(dy);

dx=dx/steps;

dy=dy/steps;

while(i++<=steps)

{

if(x>=xmin && x<=xmax && y>=ymin && y<=ymax)

{

line(x,y,x2,y2);

return;

}

x=x+dx;

y=y+dy;

}

}

void main()

{

Intn,gd,gm,x1,x2,y1,y2,xRec,yRec,b1,b2,b3,b4,l,b,yMin,yMax,xMin,xMax;

float m;

int a[10][4],i,j, flag=0,in=0;

gd=DETECT;

gm=DETECT;

initgraph(&gm,&gd,"C:\\TURBOC3\\BGI");

printf("Enter the length and breadth of the clipping window:\n");

scanf("%d%d",&l,&b);

printf("Enter the starting co-ord of the rectangle\n");

scanf("%d%d",&xRec,&yRec);

clrscr();

rectangle(xRec,yRec,xRec+l,yRec+b);

getch();

clrscr();

printf("Enter the no of lines\n");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter the co-ord of line %d\n",i+1);

for(j=0;j<2;j++)

{

scanf("%d",&a[i][j]);

}

a[i][2]=a[0][0];

a[i][3]=a[0][1];

if(i!=0)

{

a[i-1][2]=a[i][0];

a[i-1][3]=a[i][1];

}

}

clrscr();

rectangle(xRec,yRec,xRec+l,yRec+b);

for(i=0;i<n;i++)

{

line(a[i][0],a[i][1],a[i][2],a[i][3]);

}

getch();

clrscr();

rectangle(xRec,yRec,xRec+l,yRec+b);

xMin=xRec;

yMin=yRec;

xMax=xRec+l;

yMax=yRec+b;

for(i=0;i<n;i++)

{

flag=0;

x1=a[i][0];

x2=a[i][2];

y1=a[i][1];

y2=a[i][3];

if(x1>=xMin && x1<=xMax && y1>=yMin && y1<=yMax)

flag++;

if(x2>=xMin && x2<=xMax && y2>=yMin && y2<=yMax)

flag++;

switch(flag)

{

case 0:

break;

case 1: if(x2>=xMin && x2<=xMax && y2>=yMin && y2<=yMax)

{

dda(x1,y1,x2,y2,xMin,yMin,xMax,yMax);

}

else

{

dda(x2,y2,x1,y1,xMin,yMin,xMax,yMax);

}

break;

case 2: line(x1,y1,x2,y2);

break;

}

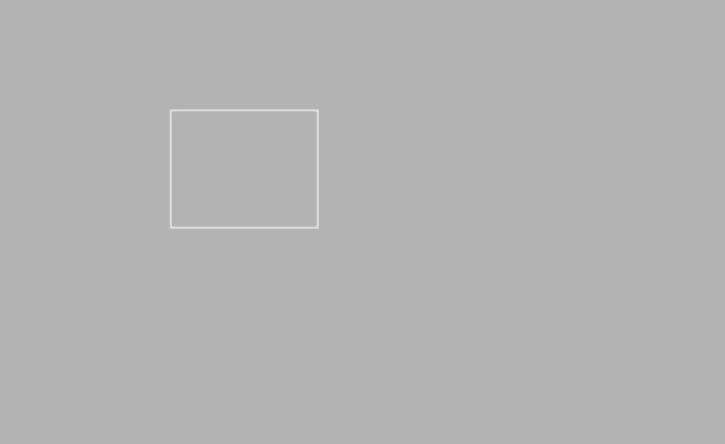
}

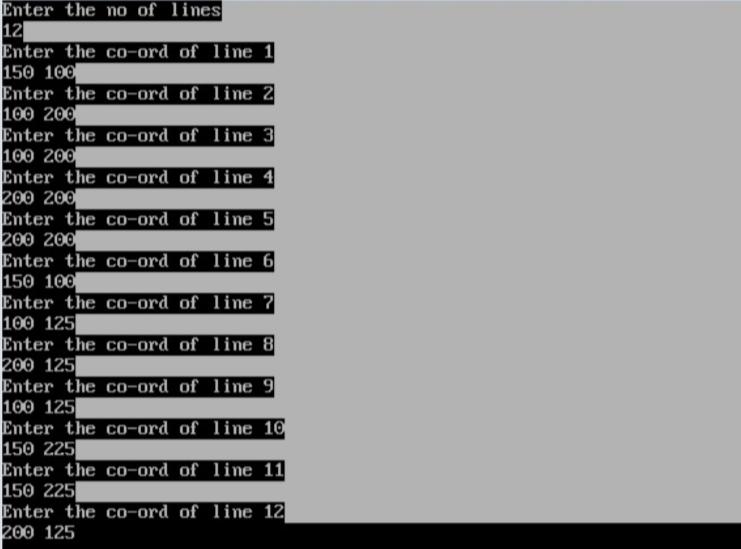
getch();

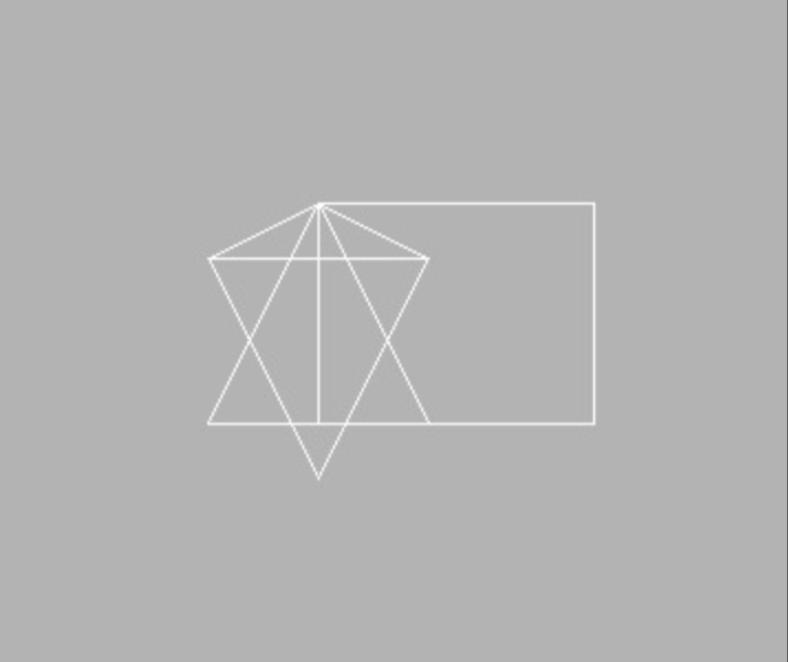
closegraph();

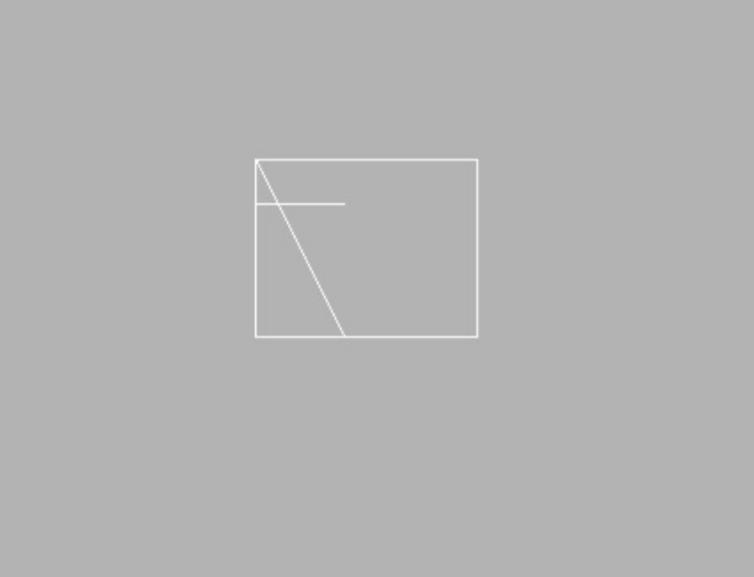
}

**OUTPUT 10**

****







# PROJECT

**TOPIC: ROCKET LAUNCHING**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void rocket();

void text();

int gd=DETECT,gm,i,j,k;

void main()

{

text();

getch();

cleardevice();

rocket();

getch();

}

void text()

{

initgraph(&gd,&gm,"C:\\Turboc3\\BGI");

settextstyle(0,EMPTY\_FILL,1);

setcolor(YELLOW);

outtextxy(50,50,"I");

outtextxy(50,60,"S");

outtextxy(50,70,"R");

outtextxy(50,80,"O");

settextstyle(0,EMPTY\_FILL,2);

setcolor(BLUE);

outtextxy(100,50,"Indian Space");

outtextxy(100,70,"Research Organisation");

settextstyle(0,EMPTY\_FILL,1);

setcolor(WHITE);

outtextxy(50,120,"WELCOME TO INDIAN SPACE REASERCH ORGANISATION.");

delay(150);

outtextxy(50,150,"TODAY 10-JULY-2015 PSLV - C28 IS READY TO LAUNCH.");

delay(150);

outtextxy(50,180,"IT CARRIES 5 SATELLITES TO DEPLOYE IN THE SUN-");

delay(150);

outtextxy(50,210,"SYNCHRONOUS ORBIT.");

delay(150);

setcolor(RED);

outtextxy(50,240,"LAUNCH MASS => 320,000 KG");

delay(150);

outtextxy(50,270,"PAYLOAD MASS => 1,440 KG");

delay(150);

outtextxy(50,300,"LAUNCH SITE => SATISH DHAWAN SPACE CENTER");

delay(150);

outtextxy(50,330,"DISTANCE TRAVELL => 647 KM");

delay(150);

outtextxy(50,360,"PAYLOAD => 3 DMCE Satellites, 1 CBNT-1 (Technology Demonstrator)");

delay(150);

outtextxy(50,390,"& 1 D-O (TD Nano Satellite)");

delay(150);

outtextxy(50,420,"SO... COUNTDOWN IS BEGIN");

delay(150);

outtextxy(50,450,"ENTER TO CONTINUE");

delay(150);

}

void rocket()

{

setcolor(BLUE);

//earth

for(i=30;i<=400;i++)

{

setcolor(BLUE);

arc(500-i,200+i,0,120,200);

arc(500-i,200+i,0,120,300);

arc(500-i,200+i,0,120,400);

arc(500-i,200+i,0,120,500);

setcolor(GREEN);

outtextxy(318,330-i,"P");

outtextxy(318,340-i,"S");

outtextxy(318,350-i,"L");

outtextxy(318,360-i,"V");

outtextxy(310,385-i,"C28");

//rocket

setcolor(WHITE );

outtextxy(10,400+i,"EARTH");

for(j=0;j<=7;j++)

{

setcolor(RED);

circle(320,300-i,0+j);

}

for(k=0;k<=15;k++)

{

setcolor(YELLOW);

circle(300,410-i,0+k);

circle(340,410-i,0+k);

circle(320,410-i,0+k);

circle(280,410-i,0+k);

circle(360,410-i,0+k);

}

setcolor(WHITE);

line(0,420+i,620,420+i);

line(340,400-i,340,320-i);

line(300,400-i,300,320-i);

line(330,370-i,330,330-i);

line(310,370-i,310,330-i);

line(310,330-i,330,330-i);

line(310,370-i,330,370-i);

line(300,380-i,340,380-i);

line(270,400-i,370,400-i);

line(270,400-i,300,380-i);

line(340,380-i,370,400-i);

line(300,320-i,340,320-i);

line(300,320-i,320,300-i);

line(340,320-i,320,300-i);

delay(50);

cleardevice();

}

setcolor(GREEN);

settextstyle(0,EMPTY\_FILL,5);

outtextxy(50,100,"CONGRATULATIONS");

outtextxy(50,200,"MISSION");

outtextxy(50,300,"SUCCESSFUL");

getch();

}

OUTPUT



