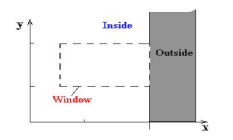
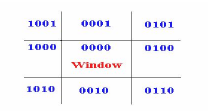
**Practical No 8.A**

**Implementation of COHEN SUTHERLAND LINE clipping algorithms.**

**Aim: Write a program to implement a Cohen Sutherland line clipping algorithm.**

**Theory:**

The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed. To determine whether endpoints are inside or outside a window, the algorithm sets up a half-space code for each endpoint. Each edge of the window defines an infinite line that divides the whole space into two half-spaces, the inside half-space and the outside half-space. As you proceed around the window, extending each edge and defining an inside half-space and an outside half-space, nine regions are created - the eight "outside" regions and the one "inside" region. Each of the nine regions associated with the window is assigned a 4-bit code to identify the region. Each bit in the code is set to either a 1(true) or a 0(false). If the region is to the left of the window, the first bit of the code is set to 1. If the region is to the top of the window, the second bit of the code is set to 1. If to the right, the third bit is set, and if to the bottom, the fourth bit is set. The 4 bits in the code then identify each of the nine regions as shown below



For any endpoint ( x , y ) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

First bit set 1: Point lies to left of window x < xmin

First bit set 1: Point lies to right of window x < xmax

First bit set 1: Point lies to below (bottom) of window y < ymin

First bit set 1: Point lies to above (top) of window y < ymax

The sequence for reading the codes' bits is LRBT (Left, Right, Bottom, Top). Once the codes for each endpoint of a line are determined, the logical AND operation of the codes determines if the line is completely outside of the window. If the logical AND of the endpoint codes is not zero, the line can be trivally rejected. For example, if an endpoint had a code of 1001 while the other endpoint had a code of 1010, the logical AND would be 1000 which indicates the line segment lies outside of the window. On the other hand, if the endpoints had codes of 1001 and 0110, the logical AND would be 0000, and the line could not be trivally rejected. The logical OR of the endpoint codes determines if the line is completely inside the window. If the logical OR is zero, the line can be trivally accepted. For example, if the endpoint codes are 0000 and 0000, the logical OR is 0000 - the line can be trivally accepted. If the endpoint codes are 0000 and 0110, the logical OR is 0110 and the line cannot be trivally accepted

**Conclusion: We have implemented Cohen Sutherland line clipping algorithm.**

**Code:**

#include<conio.h>

#include<iostream.h>

#include<graphics.h>

static int LEFT=1,RIGHT=2,BOTTOM=4,TOP=8,xl,yl,xh,yh;

int getcode(int x,int y){

int code = 0;

if(y > yh) code |=TOP;

if(y < yl) code |=BOTTOM;

if(x < xl) code |=LEFT;

if(x > xh) code |=RIGHT;

return code; }

void main() {

int gdriver = DETECT,gmode;

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

setcolor(RED);

cout<<"Enter bottom left and top right co-ordinates of window: \n";

cin>>xl>>yl>>xh>>yh;

rectangle(xl,yl,xh,yh);

int x1,y1,x2,y2;

cout<<"Enter the endpoints of the line: \n";

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

line(x1,y1,x2,y2);

getch();

int outcode1=getcode(x1,y1), outcode2=getcode(x2,y2);

int accept = 0;

while(1) {

float m =(float)(y2-y1)/(x2-x1);

if(outcode1==0 && outcode2==0){

accept = 1;

break; }

else if((outcode1 & outcode2)!=0){

break;

}else {

int x,y;

int temp;

if(outcode1==0)

temp = outcode2; else

temp = outcode1;

if(temp & TOP){

x = x1+ (yh-y1)/m;

y = yh; }

else if(temp & BOTTOM){

x = x1+ (yl-y1)/m;

y = yl;

}else if(temp & LEFT){

x = xl; y = y1+ m\*(xl-x1);

}else if(temp & RIGHT){

x = xh; y = y1+ m\*(xh-x1); }

if(temp == outcode1){

x1 = x; y1 = y;

outcode1 = getcode(x1,y1);

}else{

x2 = x; y2 = y;

outcode2 = getcode(x2,y2); } } }

setcolor(WHITE);

cout<<"After clipping:";

if(accept)

line(x1,y1,x2,y2);

getch();

closegraph(); }

**Output:**

