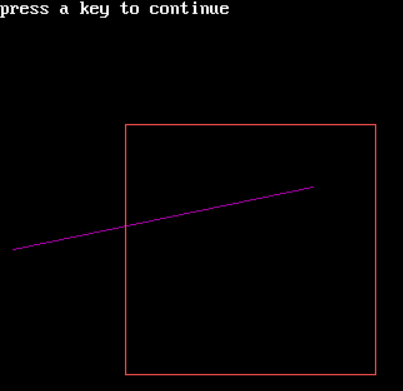
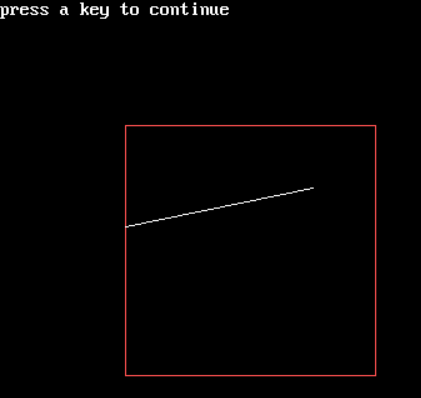
|  |  |
| --- | --- |
| **Name:** | Yash Ravindra Kerkar |
| **Roll No:** |  |
| **Class/Sem:** | SE/III |
| **Experiment No.:** | 7 |
| **Title:** | Cohen Sutherland Line Clipping Algorithm |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Marks:** |  |
| **Sign of Faculty:** |  |

# Experiment No.7

|  |  |
| --- | --- |
| **Aim** | Write a program to implement Cohen Sutherland Line Clipping Algorithm  in C. |
| **Objective** | To implement Cohen Sutherland Line Clipping Algorithm for clipping a line x1,y1 and x2, y2 with respect to a clipping window Xmin, Xmax, Ymin, Ymax. |
| **Theory** | **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 4. **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. 5. **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.   xmin,xmax are coordinates of the window. ymin,ymax are also coordinates of the window.  x1>xmax x2>xmax y1>ymax y2>ymax x1<xmin x2<xmin y1<ymin y2<ymin   1. **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. |

|  |  |
| --- | --- |
|  | The center area is having the code, 0000, i.e., region 5 is considered a rectangle window.  **Conditions Case I:**  Bitwise OR of region of 2 endpoints of line = 0000 → Line lies inside the window  **Case II :**  1. Completely invisible Bitwise OR ≠ 0000  1. Partially Visible Bitwise AND ≠ 0000  Completely invisible  **Case III :**  Partially Visible   1. Choose the endpoints of the line. 2. Find the intersection point with the window. 3. Replace the endpoints with the intersection point. |
| **Algorithm** | **Step 1:** Assign a region code for each end point.  **Step 2:** Perform Bitwise OR = 0000. Accept line & draw  **Step 3:** Perform Bitwise AND  If result ≠ 0000 then reject the line. else clip the line  → Select the endpoints.  → Find the intersection point at the window.  → Replace endpoints with the intersection pt & update the region code.  **Code:**  include <stdio.h>  #include <graphics.h>  #define INSIDE 0  #define LEFT 1  #define RIGHT 2  #define BOTTOM 4  #define TOP 8  #define X\_MIN 50  #define X\_MAX 300  #define Y\_MIN 50  #define Y\_MAX 200  int computeCode(int x, int y) {      int code = INSIDE;      if (x < X\_MIN)          code |= LEFT;      else if (x > X\_MAX)          code |= RIGHT;      if (y < Y\_MIN)          code |= BOTTOM;      else if (y > Y\_MAX)          code |= TOP;      return code;  }  void cohenSutherland(int x1, int y1, int x2, int y2) {      int code1, code2, code;      int accept = 0, done = 0;      while (!done) {          code1 = computeCode(x1, y1);          code2 = computeCode(x2, y2);          if (!(code1 | code2)) {              accept = 1;              done = 1;          } else if (code1 & code2) {              done = 1;          } else {              int x, y;              code = code1 ? code1 : code2;              if (code & TOP) {                  x = x1 + (x2 - x1) \* (Y\_MAX - y1) / (y2 - y1);                  y = Y\_MAX;              } else if (code & BOTTOM) {                  x = x1 + (x2 - x1) \* (Y\_MIN - y1) / (y2 - y1);                  y = Y\_MIN;              } else if (code & RIGHT) {                  y = y1 + (y2 - y1) \* (X\_MAX - x1) / (x2 - x1);                  x = X\_MAX;              } else if (code & LEFT) {                  y = y1 + (y2 - y1) \* (X\_MIN - x1) / (x2 - x1);                  x = X\_MIN;              }              if (code == code1) {                  x1 = x;                  y1 = y;              } else {                  x2 = x;                  y2 = y;              }          }      }      if (accept) {          int gd = DETECT, gm;          initgraph(&gd, &gm, NULL);          setcolor(WHITE);          rectangle(X\_MIN, Y\_MIN, X\_MAX, Y\_MAX);          setcolor(YELLOW);          line(x1, y1, x2, y2);          delay(5000);          closegraph();      }  }  int main() {      int x1, y1, x2, y2;      printf("Enter the starting point (x1, y1): ");      scanf("%d %d", &x1, &y1);      printf("Enter the ending point (x2, y2): ");      scanf("%d %d", &x2, &y2);      cohenSutherland(x1, y1, x2, y2);      return 0;  } |

**Output**



**Conclusion:** The Cohen-Sutherland Line Clipping Algorithm provides a simple and efficient method for clipping line segments against a rectangular window. It uses region codes to quickly determine whether an endpoint is inside or outside the window and then applies appropriate clipping operations to obtain the visible portion of the l