

## GROUP A Assignment 4 [CGT]

Name : Aditya Onkar Patel

Batch : G4 (SE4)

Roll No : 21449

Performance Date : 20/10/2021

Submission Date : 20/10/2021

TITLE: Scan Line Fill Algorithm

PROBLEM STATEMENT: Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.

## LEARNING OBJECTIVES:

To learn the concept of scan line fill algorithm and inheritance as a concept of object oriented programming in computer graphics.

LEARNING OUTCOMES: After completion of this assignment students will be able to:

1. Implement the color filling of a concave polygon with the help of scan line fill algorithm.
2. Implement the above process with the help of the concept of inheritance.

S/W AND H/W REQUIREMENTS: 64 bit open source Linux or its derivative, open source C++ programming tools like GCC, G++, Qt Creator, OpenGL etc.

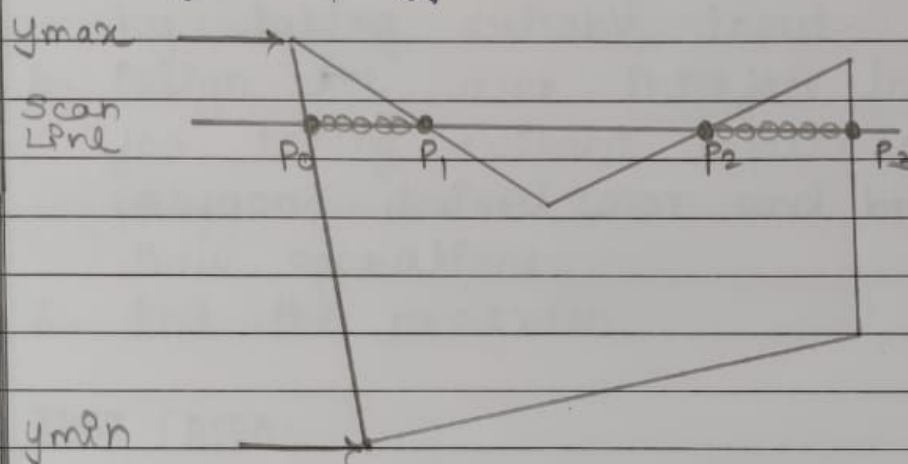
REFERENCES: 1. Programming Principles & Practice using C++, Bjaerne Stroustrup.  
2. [www.qt.io](http://www.qt.io)

### CONCEPT RELATED THEORY:

#### • SCAN LINE FILL ALGORITHM:

This algorithm works by intersecting scan line with polygon edges and fills the polygon between pairs of intersections. The following steps depict how the algorithm works.

1. Find out the  $y_{min}$  and  $y_{max}$  from the given polygon.



2. Scanline intersects with each edge of the polygon from  $y_{min}$  into  $y_{max}$ . Name each intersection point of the polygon.
3. Sort the intersection point in increasing order of  $x$  coordinate i.e.  $P_0, P_1, P_1, P_2, P_2, P_3, \dots$
4. Fill all these pairs of coordinates that are inside polygons and ignore the alternate pairs.

- **INHERITANCE:** It can be described as a process of creating new classes from existing classes. New classes inherit some of the properties and behaviour of existing classes and hence are called derived classes. Syntax:



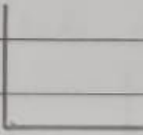
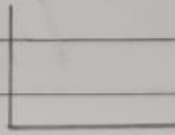


Class derived class: access specifier Base Class



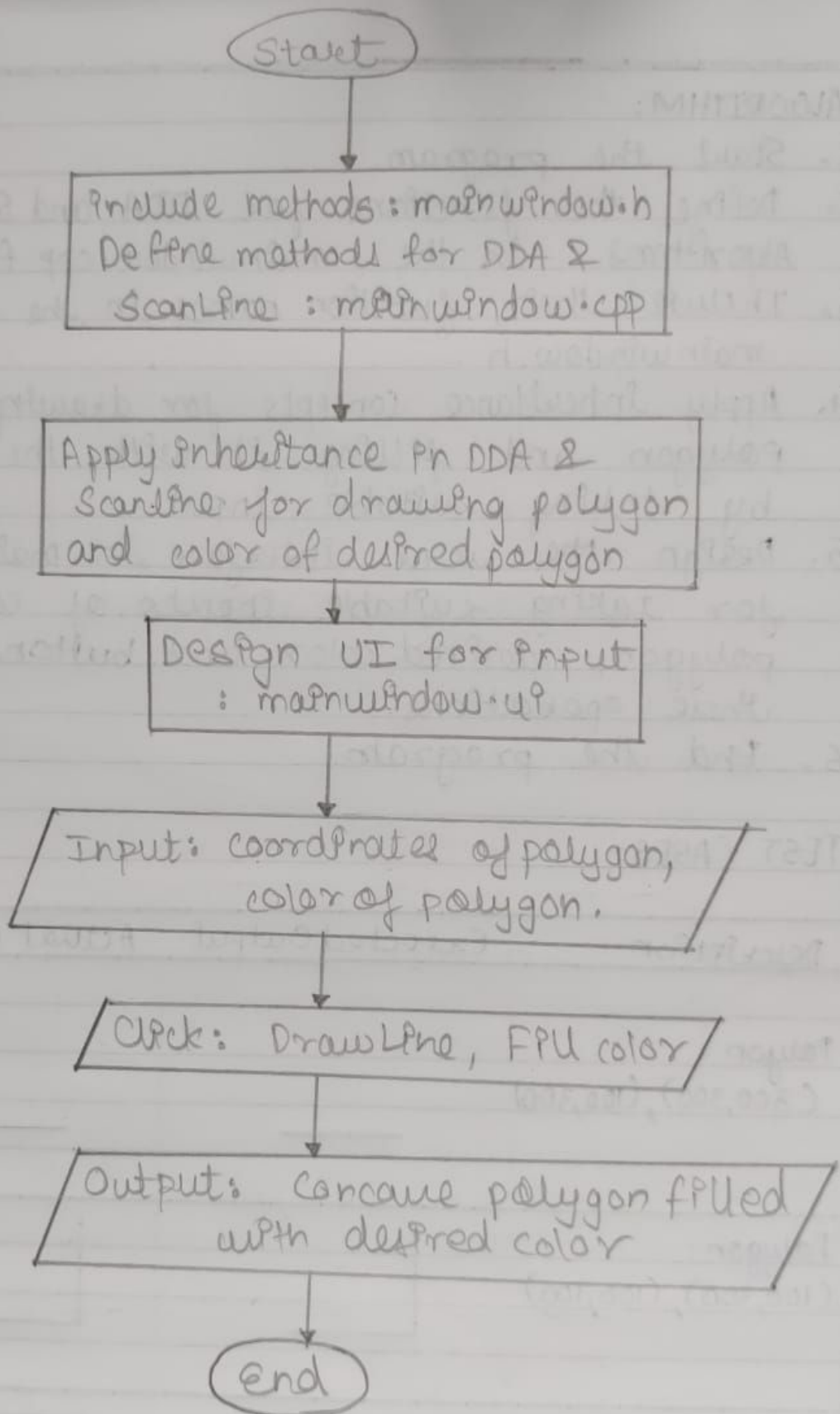
**ALGORITHM:**

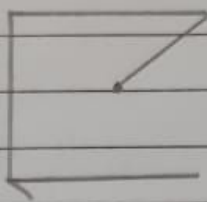
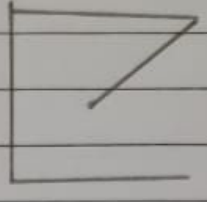
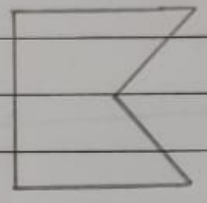
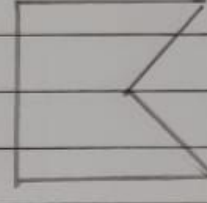

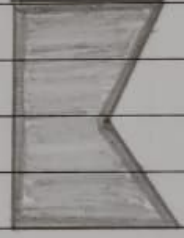
1. Start the program.
2. Define the functions for DDA and Scan Line Algorithms in the mainwindow.cpp file.
3. Include their function names in the header file mainwindow.h.
4. Apply Inheritance concepts for drawing a concave polygon and filling it with the desired color by taking suitable input.
5. Design the user interface in mainwindow.ui for taking suitable inputs of coordinates of polygon, desired color and buttons for executing these operations.
6. End the program.

**TEST CASES:**

Sr.No.	Description	Expected Output	Actual Output	Status
1.	Polygon : (300,300), (100,300)			Pass
2.	Polygon : (100,300), (100,100)			Pass
3.	Polygon : (100,100), (300,100)			Pass

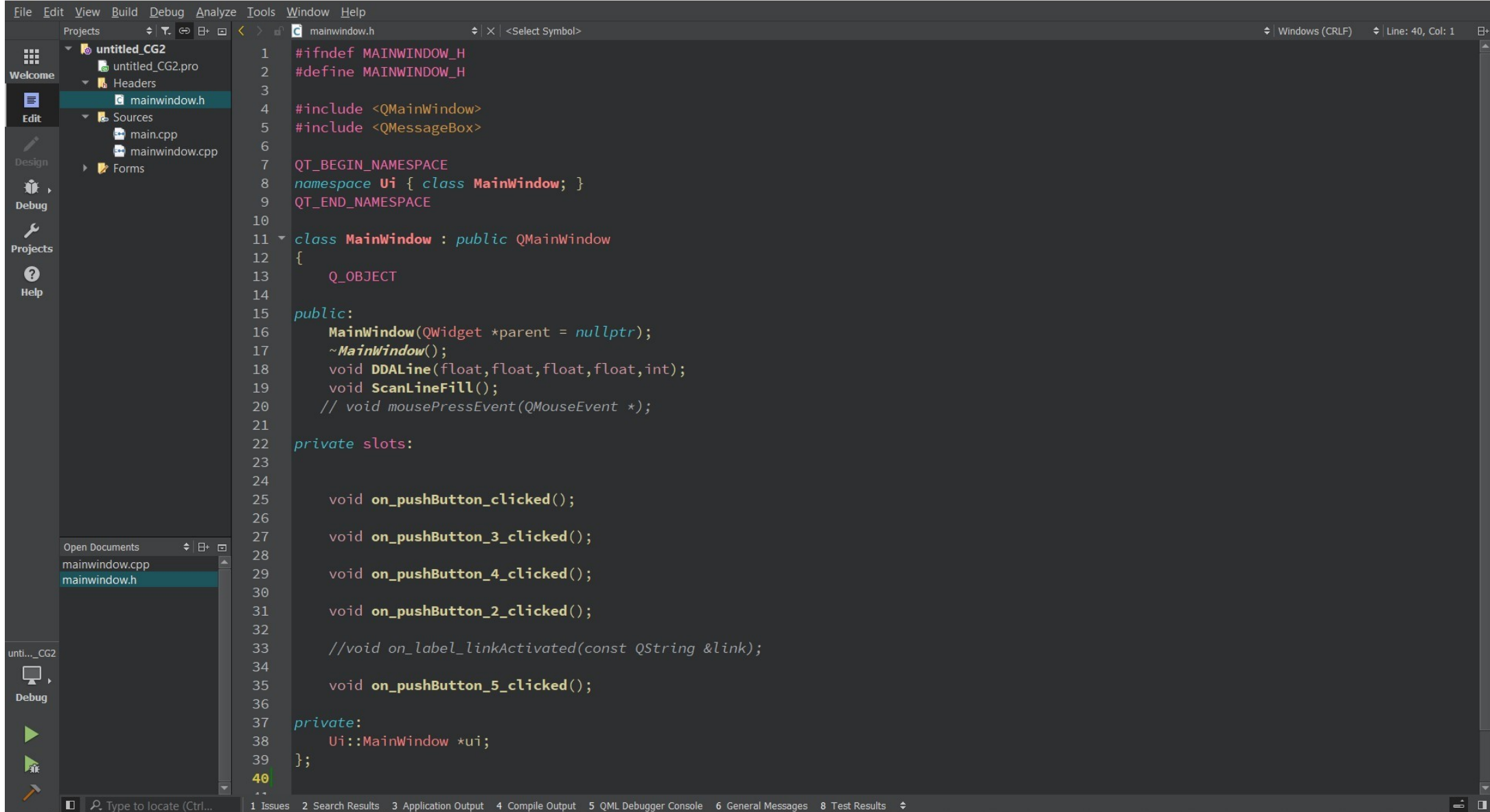
# FLOW CHART:



Sr.No.	Description	Expected Output	Actual Output	Status
4.	Polygon: (300,100), (200,200)			Pass
5.	Polygon: (200,200), (300,300)			Pass
6.	FPU Polygon: Select color & Click Fill			Pass

### CONCLUSION:

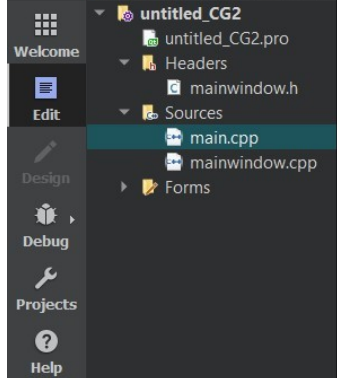
The concepts of Scan Line Fill Algorithm for colouring concave polygons were successfully understood and implemented using concepts of inheritance.



```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QMessageBox>
6
7  QT_BEGIN_NAMESPACE
8  namespace Ui { class MainWindow; }
9  QT_END_NAMESPACE
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18     void DDALine(float,float,float,float,int);
19     void ScanLineFill();
20     // void mousePressEvent(QMouseEvent *);
21
22 private slots:
23
24
25     void on_pushButton_clicked();
26
27     void on_pushButton_3_clicked();
28
29     void on_pushButton_4_clicked();
30
31     void on_pushButton_2_clicked();
32
33     //void on_label_linkActivated(const QString &link);
34
35     void on_pushButton_5_clicked();
36
37 private:
38     Ui::MainWindow *ui;
39 };
40
```

1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 6 General Messages 8 Test Results





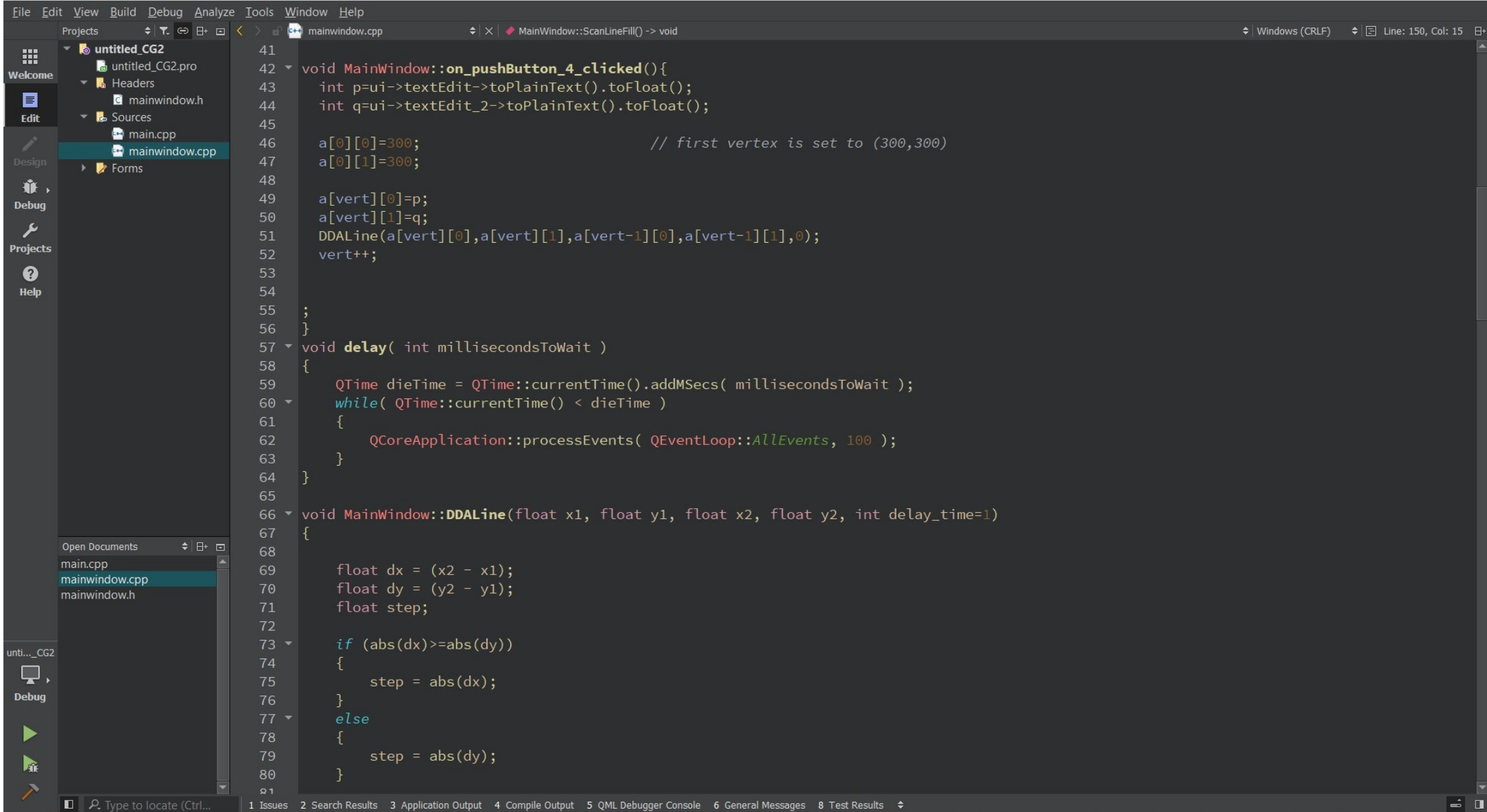
```
1 #include "mainwindow.h"
2
3 #include <QApplication>
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     MainWindow w;
9     w.show();
10    return a.exec();
11 }
12
```

The image shows the Qt Creator IDE with the following components:

- Left Sidebar:** Contains icons for Welcome, Edit, Design, Debug, and Projects. Below these is a list of open documents: main.cpp, mainwindow.cpp (selected), and mainwindow.h.
- Top Menu Bar:** File, Edit, View, Build, Debug, Analyze, Tools, Window, Help.
- Top Toolbar:** Includes icons for file operations and a status bar showing "Line: 150, Col: 15".
- Editor:** Displays the C++ code for mainwindow.cpp. The code includes headers, defines window dimensions, and implements the MainWindow constructor and destructor.
- Bottom Status Bar:** Contains tabs for 1 Issues, 2 Search Results, 3 Application Output, 4 Compile Output, 5 QML Debugger Console, 6 General Messages, and 8 Test Results.

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3 #include <QColorDialog>
4 #include <math.h>
5 #include <QTime>
6 #define height 500
7 #define width 500
8 using namespace std;
9
10 int vert,i,y,xi[20],yi[20],j,temp,k;
11 int a[20][2];
12 float slope[20],dx,dy;
13
14
15
16 QImage img(width, height, QImage::Format_RGB888);
17 QRgb rgb(qRgb(255,255,255));
18
19 MainWindow::MainWindow(QWidget *parent)
20     : QMainWindow(parent)
21     , ui(new Ui::MainWindow)
22 {
23     ui->setupUi(this);
24
25     for (int x = 0; x < width; ++x)
26     {
27         for (int y = 0; y < height; ++y)
28         {
29             img.setPixel(x, y, qRgb(0, 0, 0)); //setting all the pixels on the screen to black (0,0,0)
30         }
31     }
32     ui->label->setPixmap(QPixmap::fromImage(img));
33     vert=1;
34 }
35
36
37 MainWindow::~MainWindow()
38 {
39     delete ui;
40 }
```





The screenshot shows the Qt Creator IDE with a C++ project named 'untitled\_CG2'. The main window displays the code for 'mainwindow.cpp'. The code defines a 'MainWindow' class with a 'ScanLineFill()' method and a 'DDALine()' static method. The 'ScanLineFill()' method uses a vertical scanline algorithm to fill a polygon by iterating over vertices and drawing horizontal lines. The 'DDALine()' method implements a Digital Differential Analyzer (DDA) algorithm for drawing a line between two points (x1, y1) and (x2, y2) with a specified delay time.

```
41
42 void MainWindow::on_pushButton_4_clicked() {
43     int p=ui->textEdit->toPlainText().toFloat();
44     int q=ui->textEdit_2->toPlainText().toFloat();
45
46     a[0][0]=300; // first vertex is set to (300,300)
47     a[0][1]=300;
48
49     a[vert][0]=p;
50     a[vert][1]=q;
51     DDALine(a[vert][0],a[vert][1],a[vert-1][0],a[vert-1][1],0);
52     vert++;
53
54
55 ;
56 }
57 void delay( int millisecondsToWait )
58 {
59     QTime dieTime = QTime::currentTime().addMSecs( millisecondsToWait );
60     while( QTime::currentTime() < dieTime )
61     {
62         QApplication::processEvents( QEventLoop::AllEvents, 100 );
63     }
64 }
65
66 void MainWindow::DDALine(float x1, float y1, float x2, float y2, int delay_time=1)
67 {
68
69     float dx = (x2 - x1);
70     float dy = (y2 - y1);
71     float step;
72
73     if (abs(dx)>=abs(dy))
74     {
75         step = abs(dx);
76     }
77     else
78     {
79         step = abs(dy);
80     }
81 }
```

The interface includes a sidebar with project files (main.cpp, mainwindow.h, mainwindow.cpp), a toolbar with icons for Welcome, Edit, Design, Debug, and Projects, and a bottom status bar with tabs for Issues, Search Results, Application Output, Compile Output, QML Debugger Console, General Messages, and Test Results.

mainwindow.cpp @ untitled\_CG2 - Qt Creator

The image shows a Qt Creator IDE interface. The top menu bar includes File, Edit, View, Build, Debug, Analyze, Tools, Window, and Help. The left sidebar contains a 'Projects' panel showing a project named 'untitled\_CG2' with sub-items 'untitled\_CG2.pro', 'Headers', 'mainwindow.h', 'Sources', 'main.cpp', and 'mainwindow.cpp'. Below this is an 'Open Documents' panel listing 'main.cpp', 'mainwindow.cpp', and 'mainwindow.h'. The main editor area displays the code in 'mainwindow.cpp'. The code includes a 'delay(delay\_time);' statement, followed by a 'void MainWindow::ScanLineFill()' function. This function contains logic for calculating the slope of a line segment and filling it with pixels. The code is as follows:

```
197     delay(delay_time);
198 }
199
200 }
201
202 void MainWindow::ScanLineFill()
203 {
204     int ymax=0,ymin=100000000;
205     a[0][0]=300;
206     a[0][1]=300;
207     a[vert][0]=a[0][0];
208     a[vert][1]=a[0][1];
209     for(i=0;i<vert-1;i++)
210     {
211         if(a[i][1]>ymax){
212             ymax=a[vert][i];
213         }
214         if(a[i][1]<ymin){
215             ymin=a[vert][i];
216         }
217     }
218     for(i=0;i<vert;i++){
219         dx=a[i+1][0]-a[i][0];
220         dy=a[i+1][1]-a[i][1];
221         if(dx==0.0){slope[i]=1.0;}
222         if(dy==0.0){slope[i]=0.0;}
223         if(dx!=0.0 and dy!=0.0){slope[i]= float(dx/dy);}
224     }
225     for(int y=0;y<500;y++){
226         int k=0;
227         for(i=0;i<vert;i++){
228             if((y>=a[i][1] and y<a[i+1][1])or (y>=a[i+1][1]and y<a[i][1])){
229                 xi[k]= a[i][0]+(slope[i]*(y-a[i][1]));
230                 yi[k]= a[i][1];
231                 k++;
232             }
233         }
234     }
235     for(i=0;i<k-1;i++){
236         for(j=0;j<k-1-i;j++){
237             if(a[i][1]<=a[i+1][1] and a[i+1][1]<=a[i+2][1]){
```

The bottom status bar shows 'Line: 150, Col: 15' and a list of tabs: 1 Issues, 2 Search Results, 3 Application Output, 4 Compile Output, 5 QML Debugger Console, 6 General Messages, 8 Test Results.

```
File Edit View Build Debug Analyze Tools Window Help
Projects untitled_CG2
  untitled_CG2.pro
  Headers
  Sources
  mainwindow.h
  mainwindow.cpp
  Forms
  Forms

Open Documents
main.cpp
mainwindow.cpp
mainwindow.h

1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 6 General Messages 8 Test Results

137         if(xi[j]>xi[j+1]){
138             temp=xi[j];
139             xi[j]=xi[j+1];
140             xi[j+1]=temp;
141         }
142     }
143 }
144 for(j=1;j<k;j++){
145     DDALine(xi[j],y,a[j][0],y);
146
147     //for(int z=1;z<k;z++) {
148     //    DDALine(xi[j],yi[z],xi[j],yi[z]);
149
150     //}
151 }
152 }
153 }
154 }
155
156
157 void MainWindow::on_pushButton_clicked() //COLOR
158 {
159     QColor color(QColorDialog::getColor().rgb());
160     rgb = color;
161 }
162
163
164 void MainWindow::on_pushButton_3_clicked()
165 {
166     QMessageBox message;
167     message.information(0,"Screen has been cleared.", "Please draw new polygon"); //RESET
168     for (int x = 0; x < width; ++x)
169     {
170         for (int y = 0; y < height; ++y)
171         {
172             img.setPixel(x, y, QColor(0, 0, 0));
173         }
174     }
175
176     ui->label->setPixmap(QPixmap::fromImage(img));
177 }
```

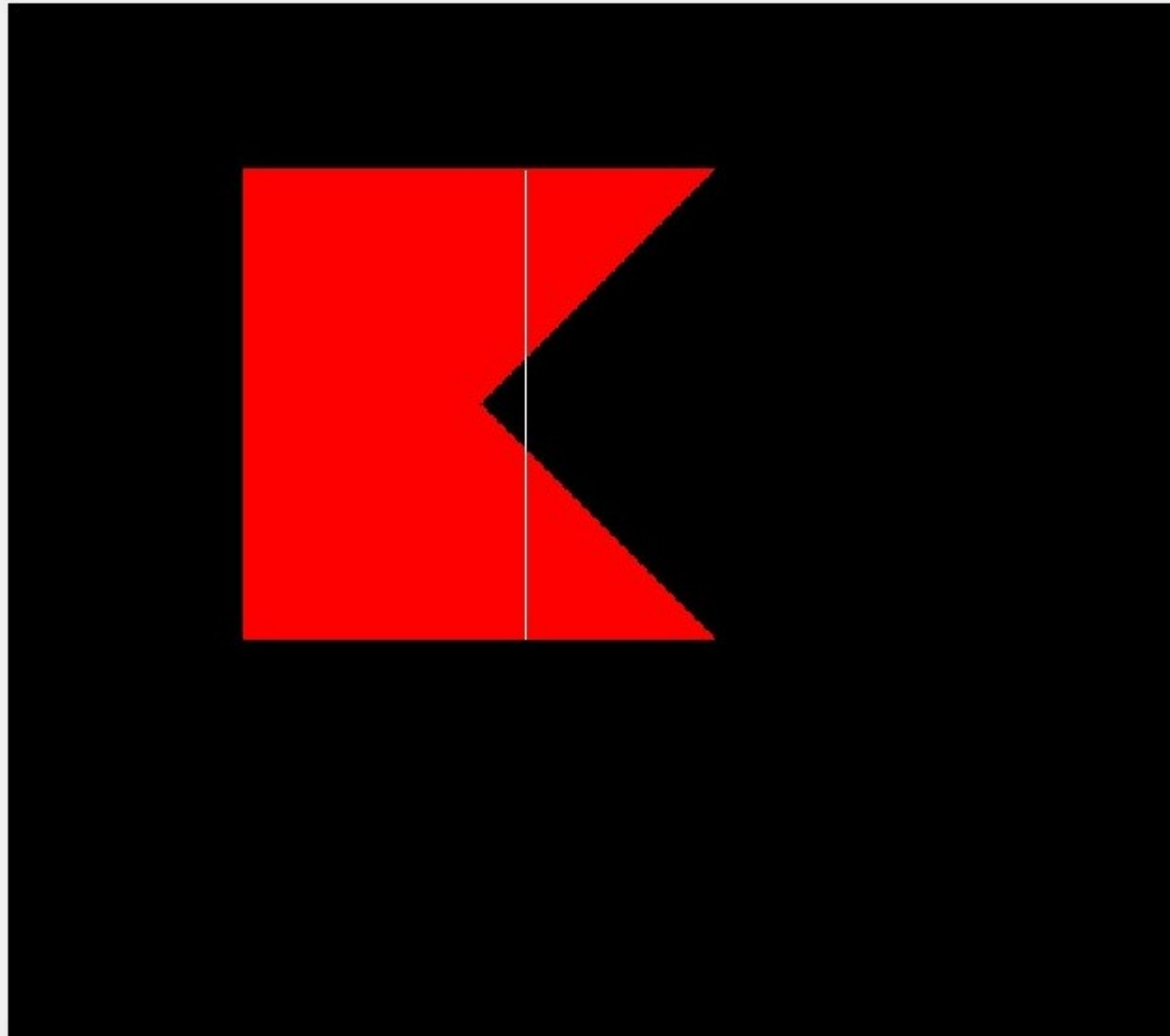


The screenshot shows the Qt Creator IDE with a C++ project named 'untitled\_CG2'. The left sidebar contains a 'Projects' panel showing the project structure: 'untitled\_CG2' (containing 'untitled\_CG2.pro'), 'Headers' (containing 'mainwindow.h'), 'Sources' (containing 'main.cpp' and 'mainwindow.cpp'), and 'Forms'. Below this is an 'Open Documents' panel showing 'main.cpp', 'mainwindow.cpp' (selected), and 'mainwindow.h'. The main editor window displays the code for 'mainwindow.cpp', with the current function being 'MainWindow::ScanLineFill()' at line 150, column 15. The code implements a scanline fill algorithm for a polygon. It starts by setting the pixel color to black (0, 0, 0) for each pixel in the current scanline. Then, it iterates over the scanlines (q) and for each scanline, it iterates over the pixels (b) and sets the pixel color to black. The algorithm also checks for a warning message if the polygon has not been closed properly. The status bar at the bottom shows the current line and column, and a list of tabs for 'Issues', 'Search Results', 'Application Output', 'Compile Output', 'QML Debugger Console', 'General Messages', and 'Test Results'.

```
169     {
170         for (int y = 0; y < height; ++y)
171         {
172             img.setPixel(x, y, qRgb(0, 0, 0));
173         }
174     }
175
176
177     ui->label->setPixmap(QPixmap::fromImage(img));
178     for (int q = 0; q <= vert; q++)
179     {
180         for (int b=0; b<2; b++)
181         {
182             a[q][b]=0;
183         }
184     }
185     vert=1;
186
187 }
188
189 void MainWindow::on_pushButton_2_clicked()           //FILL
190 {
191     QMessageBox message;
192     if(a[0][0]==a[vert-1][0] && a[0][1]==a[vert-1][1])
193     {
194         ScanLineFill();
195         for (int q = 0; q <= vert; q++)
196         {
197             for (int b=0; b<2; b++)
198             {
199                 a[q][b]=0;
200             }
201         }
202         vert=1;
203     }
204     else
205     {
206         message.information(0,"Warning!","The polygon has not been closed properly! Please check the coordinates!");
207     }
208 }
```

vik[a[i][1]]:

MainWindow



x

y

COLOR

FILL

CLEAR

300



300



Scan Line

Accept