Actual Code students should work towards developing.
SUGGESTED PROGRAMMING SOLUTION ALGORITHM

```
#include "imageloader.h"
#include <QtGui>
//#include <QString>
#include <cmath>
//#include <QVector>

#define PI 3.14159265358979353
#define SD .4
#define SCALE 1.25


int imageLoader::finalValue(double currentElement, double CDF1, int newRange, double CDF_Final)
{
        return int(((currentElement - CDF1)/(CDF_Final - CDF1))*(newRange-1) + .5);
}
imageLoader::imageLoader(QWidget *parent):
QLabel(parent)
{

  setAutoFillBackground(true);
  //setMinimumWidth(700);
  //setMinimumHeight(700);
  setPalette(QPalette(QColor(Qt::black)));

}
void imageLoader::loadBackImage(QPixmap backPixmap)
{

        m_backImage = backPixmap.toImage();

}
void imageLoader::loadFrontImage(QPixmap frontPixmap)
{
        m_frontImage = frontPixmap.toImage();


}
double imageLoader::FRX(int x, int height, double std)
{
   double RX =-1+(2*(double)x)/((double)height-1);

        return SCALE*(RX/std);
```

```
}
double imageLoader::CDF(double x)
        {
                const double b1 =  0.319381530;
                const double b2 = -0.356563782;
                const double b3 =  1.781477937;
                const double b4 = -1.821255978;
                const double b5 =  1.330274429;
                const double p  =  0.2316419;
                const double c  =  0.39894228;

                if(x >= 0.0) {
                        double t = 1.0 / ( 1.0 + p * x );
                        return (1.0 - c * exp( -x * x / 2.0 ) * t *
                ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
        }
                else {
                        double t = 1.0 / ( 1.0 - p * x );
                        return ( c * exp( -x * x / 2.0 ) * t *
                        ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
                }
        }
QPixmap imageLoader::mergeImages()
{
        float frontFocalLength = 300.0;
        float backFocalLength = 70.0;
        double focalRatio = frontFocalLength/backFocalLength;


        QImage m_mergedImage(m_backImage.width()*focalRatio,m_backImage.height()*focalRatio,
QImage::Format_RGB32);
   //QImage m_warpedImage(1200,800, QImage::Format_RGB32);

        int x_offSet = (m_backImage.width()*focalRatio - m_frontImage.width())/2 + 15;
        int y_offSet = (m_backImage.height()*focalRatio - m_frontImage.height())/2 - 10;
        m_mergedImage = m_frontImage.scaledToWidth(m_frontImage.width()*focalRatio);

        for(int x=1;x<m_frontImage.width()-1;x++){
                for(int y=1;y<m_frontImage.height()-1;y++){
                        m_mergedImage.setPixel(x+ x_offSet,y+ y_offSet,m_backImage.pixel(x,y));
                }
        }

        int mergedHeight = m_mergedImage.height();
        int mergedWidth = m_mergedImage.width();
```

```
        int finalWidth = 800;
        int finalHeight = 600;
double CDF1Y = CDF(FRX(1,mergedHeight,SD));
        double CDF1X = CDF(FRX(1,mergedWidth,SD));
        double CDFHeight = CDF(FRX(mergedHeight,mergedHeight,SD));
        double CDFWidth = CDF(FRX(mergedWidth,mergedWidth,SD));

        QVector<int> xMap;
        QVector<int> yMap;

        xMap.resize(mergedWidth);
        yMap.resize(mergedHeight);

        for(int x=0; x<xMap.size(); x++){
                xMap[x] = finalValue(CDF(FRX(x,mergedWidth,SD)),CDF1X,finalWidth, CDFWidth);
        }
        for(int y=0; y<yMap.size(); y++){
                yMap[y] = finalValue(CDF(FRX(y,mergedHeight,SD)),CDF1Y,finalHeight, CDFHeight);
        }
        QVector<QVector<QList<QRgb>>> imageMatrix;
        imageMatrix.resize(xMap.size());
        for(int x=0;x<imageMatrix.size(); x++){
                imageMatrix[x].resize(yMap.size());
        }
        for(int x=0; x<xMap.size();x++){
                for(int y=0; y<yMap.size(); y++){
                        imageMatrix[xMap[x]][yMap[y]].push_front(m_mergedImage.pixel(x,y));
                }
        }
        QVector<QVector<QRgb>> finalImageMatrix;
        finalImageMatrix.resize(xMap.size());
        for(int x=0;x<finalImageMatrix.size();x++){
                finalImageMatrix[x].resize(yMap.size());
        }
        /*int topPixelRed = 0;
        int topPixelGreen = 0;
        int topPixelBlue = 0;
        int bottomPixelRed = 0;
        int bottomPixelGreen = 0;
        int bottomPixelBlue = 0;
        int leftPixelRed = 0;
        int leftPixelGreen = 0;
        int leftPixelBlue = 0;
        int rightPixelRed = 0;
        int rightPixelGreen = 0;
```

```
        int rightPixelBlue = 0;
        int finalRed = 0;
        int finalGreen = 0;
        int finalBlue = 0;
        int mySize = 0;
        //QRgb myColor = 0;
        for(int x=1; x<finalImageMatrix.size()-1;x++){
                for(int y=1;y<finalImageMatrix[x].size()-1;y++){
                        mySize = imageMatrix[x][y].size();
                        qDebug()<<"Wompass"<<"x="<<x<<" y="<<y<<" and mySize="<<mySize;
                        switch(mySize){
                                case 0:
                                        topPixelRed = qRed(imageMatrix[x][y-1][0]);
                                        topPixelGreen = qGreen(imageMatrix[x][y-1][0]);
                                        topPixelBlue = qBlue(imageMatrix[x][y-1][0]);
                                        bottomPixelRed = qRed(imageMatrix[x][y+1][0]);
                                        bottomPixelGreen = qGreen(imageMatrix[x][y+1][0]);
                                        bottomPixelBlue = qBlue(imageMatrix[x][y+1][0]);
                                        leftPixelRed = qRed(imageMatrix[x-1][y][0]);
                                        leftPixelGreen = qGreen(imageMatrix[x-1][y][0]);
                                        leftPixelBlue = qBlue(imageMatrix[x-1][y][0]);
                                        rightPixelRed = qRed(imageMatrix[x+1][y][0]);
                                        rightPixelGreen = qGreen(imageMatrix[x+1][y][0]);
                                        rightPixelBlue = qBlue(imageMatrix[x+1][y][0]);
                                        finalRed = (topPixelRed + bottomPixelRed + leftPixelRed +
rightPixelRed)/4;
                                        finalGreen = (topPixelGreen + bottomPixelGreen + leftPixelGreen +
rightPixelGreen)/4;
                                        finalBlue = (topPixelBlue + bottomPixelBlue + leftPixelBlue +
rightPixelBlue)/4;
                                        finalImageMatrix[x][y]=(qRgb(finalRed,finalGreen,finalBlue));
                                        qDebug()<<"Wompass"<<"x="<<x<<" y="<<y<<" and
mySize="<<mySize;
                                        break;
                                case 1:
                                        finalImageMatrix[x][y]=imageMatrix[x][y][0];
                                        qDebug()<<"Wompass"<<"x="<<x<<" y="<<y<<" and
mySize="<<mySize;
                                        break;
                                default:
                                        for(int t=0;t<mySize-1;t++){
                                                finalRed += qRed(imageMatrix[x][y][t]);
                                                finalGreen += qGreen(imageMatrix[x][y][t]);
                                                finalBlue += qBlue(imageMatrix[x][y][t]);
                                        }
                                        qDebug()<<"Wompass"<<"x="<<x<<" y="<<y<<" and
mySize="<<mySize;
```

```
                                        finalRed = finalRed/mySize;
                                        finalGreen = finalGreen/mySize;
                                        finalBlue = finalBlue/mySize;
                                        finalImageMatrix[x][y]= qRgb(finalRed,finalGreen,finalBlue);
                                        break;
                                }
                        }
                }*/
                QImage testImage(xMap.size(),yMap.size(), QImage::Format_RGB32);
                for(int x = 0; x<finalImageMatrix.size() ; x++){
                        for (int y = 0; y<finalImageMatrix[x].size(); y++){
                                testImage.setPixel(x,y,finalImageMatrix[x][y]);
                        }
                }
        QImage finalImage(finalWidth,finalHeight, QImage::Format_RGB32);
                for(int x = 0; x<mergedWidth ; x++){
                        for (int y = 0; y<mergedHeight; y++){
                                finalImage.setPixel(xMap[x],yMap[y], m_mergedImage.pixel(x,y));
                        }
                }
                qDebug()<< "imageMatrix[1][114]="<<imageMatrix[1][114].size();
                qDebug()<<"finalImageMatrix.size()="<<finalImageMatrix.size()<<"and
finalImageMatrix[x].size()="<<finalImageMatrix[1].size();
                return QPixmap().fromImage(finalImage);


}

//imageLoader.h


#ifndef IMAGELOADER_H

#define IMAGELOADER_H


#include <QLabel>

#include <QString>

#include <QColor>


class QImage;
```

```
class QColor;


class imageLoader : public QLabel
{
        Q_OBJECT


public:
        imageLoader(QWidget* parent=0);
    void loadBackImage(QPixmap backPixmap);
        void loadFrontImage(QPixmap frontPixmap);
        QPixmap mergeImages();
        double CDF(double x);
        double FRX(int x, int height, double std);
        int finalValue(double currentElement, double CDF1, int newRange, double CDF_Final);


private:
        QImage m_backImage;
        QImage m_frontImage;
        QImage m_mergedImage;
        QString myString;
        QColor myColor;


};
#endif



//main.cpp
```

```
#include <QApplication>
#include "testWindow.h"

int main(int argc, char ** argv){

        QApplication app(argc, argv);
        testWindow* test = new testWindow();
    test->show();
        return(app.exec());
}


//testWindow.cpp
#include "testWindow.h"
#include <QtGui>


testWindow::testWindow()
{

         m_pBackImage = new QLabel;
         backImageLabel = new QLabel;
         m_pFrontImage = new QLabel;
         frontImageLabel = new QLabel;
         m_pImageLoader = new imageLoader;
         m_pBackImage->setPixmap(QPixmap("backImage.png"));
         m_pFrontImage->setPixmap(QPixmap("frontImage.png"));
```

```
        frontImageLabel->setText("300mm Focal Length");
        frontImageLabel->setScaledContents(true);
        m_pImageLoader->setScaledContents(true);
        m_pImageLoader->loadBackImage(QPixmap("frontImage.png"));
        backImageLabel->setText("70mm Focal Length");
        m_pImageLoader->loadFrontImage(QPixmap("backImage.png"));
        QPixmap dummy = m_pImageLoader->mergeImages();
        m_pImageLoader->setPixmap(dummy);
    m_pImageLoader->resize(dummy.size());
        QVBoxLayout* vLayout = new QVBoxLayout();
        vLayout->addWidget(frontImageLabel);
        vLayout->addWidget(m_pFrontImage);
        vLayout->addWidget(backImageLabel);
        vLayout->addWidget(m_pBackImage);

        QHBoxLayout* hLayout = new QHBoxLayout();
        hLayout->addWidget(m_pImageLoader);
        hLayout->addLayout(vLayout);

    QWidget* layoutWidget = new QWidget(this);
        layoutWidget->setLayout(hLayout);

        setCentralWidget(layoutWidget);
};

//testWindow.h
```

```
#ifndef TESTWINDOW_H

#define TESTWINDOW_H


#include "../../imageLoaderWidget/imageloader.h"

#include<QMainWindow>

class QLabel;

//class imageLoader;


class testWindow : public QMainWindow

{

        Q_OBJECT


public:

        testWindow();

private:

        imageLoader* m_pImageLoader;

        QLabel* m_pFrontImage;

        QLabel* m_pBackImage;

        QLabel* frontImageLabel;

        QLabel* backImageLabel;

};

#endif


//imageLoader.pro

TEMPLATE   = lib

CONFIG  += qt warn_on no_keywords staticlib

QT       +=
```

```
TARGET     = imageLoaderWidget

SOURCES          = imageloader.cpp

HEADERS          = imageloader.h

LIBS     +=



# Treat warnings as errors

win32:QMAKE_CXXFLAGS += /WX


CONFIG(debug, debug|release){

          # Debug build options

}

else{

          # Release build options

}


//MainProject.pro

TEMPLATE   = app

CONFIG  += qt warn_on no_keywords embed_manifest_exe

QT       +=

TARGET     = RETProjectGonyea2008

SOURCES          = main.cpp testWindow.cpp

HEADERS          = testWindow.h

LIBS     +=



# Treat warnings as errors
```

```
win32:QMAKE_CXXFLAGS += /WX


CONFIG(debug, debug|release){
        # Debug build options
        # Enable a read-only console window (i.e. for printf etc.)
          CONFIG   += console
        LIBS += ../../RETProjectGonyea2008/imageLoaderWidget/Debug/imageLoaderWidget.lib
        POST_TARGETDEPS += $$LIBS


}
else{
        # Release build options
        # Enable a read-only console window (i.e. for printf etc.)
        # CONFIG   += console
        LIBS += ../../RETProjectGonyea2008/imageLoaderWidget/release/imageLoaderWidget.lib
        POST_TARGETDEPS += $$LIBS
}
```