**Name:** *Matthew Chu*

**Program:** *Computer Engineering Technology*

**Year:** *1st Year*

**Assignment:** *Design Project #1*

**Problem Statement:**

Create a C++ program that can rotate an image a certain number of degrees as specified by the user. The length and width of the image will be provided by the user, as well as the angle at which the user wishes to rotate the image. The program will eventually display the original coordinates and the new, rotated coordinates.

**Requirements Analysis:**

Seven variables will be declared to hold the seven grade elements. The program will identify where the future additional code for user input will be placed. The program will calculate the average for the course. The program will output the calculated average to the console, no permanent record of the average will be created. In the class the midterm and final are 15% of the final grade, 3 problems are 30% and 2 projects are 40% of the final average.

Variables must be declared to hold the coordinates of the image. The program will identify the information that the user must input. The program will then calculate the new coordinates. The program will output the original coordinates, then the new coordinates, after the image has been rotated.

INPUTS: length of image, width of image, angle to be rotated

OUTPUTS: the original coordinates of the image and the new coordinates of the image

SPECIAL FACTORS: Prepare for future enhancements that will allow the user to input the length, width, and height

UNRESOLVED ISSUES: none

**Program File(s):**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Title: ImageRotation\_mc.cpp \*\*\*/

/\*\*\* Course: Computational Problem Solving CPET-121 \*\*\*/

/\*\*\* Developer: Matthew Chu \*\*\*/

/\*\*\* Data: Feb 1, 2018 \*\*\*/

/\*\*\* Description: Image Rotation \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** <iostream>

**#include** <cmath>

**#define** \_USE\_MATH\_DEFINES

**#include** <math.h>

**#include** <stdio.h>

**#include** <iomanip>

**using** **namespace** std;

**int** **main**(){

**float** height, width, angle;

cout << "Enter the Width of the Image: ";

cin >> width;

cout << "Enter the Height of the Image: ";

cin >> height;

cout << "Enter the Angle of Rotation: ";

cin >> angle;

//bottom left corner

**float** Cx0 = 0;

**float** Cy0 = 0;

//bottom right corner

**float** Cx1 = width;

**float** Cy1 = 0;

//top right corner

**float** Cx2 = width;

**float** Cy2 = height;

//top left corner

**float** Cx3 = 0;

**float** Cy3 = height;

//temp vars to store converted coord

**float** tmpx0, tmpy0, tmpx1, tmpy1, tmpx2, tmpy2, tmpx3, tmpy3;

**float** Dx0, Dy0, Dx1, Dy1, Dx2, Dy2, Dx3, Dy3;

//rect to polar conversion

Dx0 = **sqrt**(pow(Cx0,2.0)+pow(Cy0,2.0));

Dy0 = atan2(Cy0, Cx0)\*180.0/M\_PI;

Dx1 = **sqrt**(pow(Cx1,2.0)+pow(Cy1,2.0));

Dy1 = atan2(Cy1, Cx1)\*180.0/M\_PI;

Dx2 = **sqrt**(pow(Cx2,2.0)+pow(Cy2,2.0));

Dy2 = atan2(Cy2, Cx2)\*180.0/M\_PI;

Dx3 = **sqrt**(pow(Cx3,2.0)+pow(Cy3,2.0));

Dy3 = atan2(Cy3, Cx3)\*180.0/M\_PI;

//store polar coord

tmpx0 = Dx0;

tmpy0 = Dy0;

tmpx1 = Dx1;

tmpy1 = Dy1;

tmpx2 = Dx2;

tmpy2 = Dy2;

tmpx3 = Dx3;

tmpy3 = Dy3;

//rewrite vars to hold rect coord

Dx0 = tmpx0\***cos**(((tmpy0+angle)\*M\_PI/180.0));

Dy0 = tmpx0\***sin**(((tmpy0+angle)\*M\_PI/180.0));

Dx1 = tmpx1\***cos**(((tmpy1+angle)\*M\_PI/180.0));

Dy1 = tmpx1\***sin**(((tmpy1+angle)\*M\_PI/180.0));

Dx2 = tmpx2\***cos**(((tmpy2+angle)\*M\_PI/180.0));

Dy2 = tmpx2\***sin**(((tmpy2+angle)\*M\_PI/180.0));

Dx3 = tmpx3\***cos**(((tmpy3+angle)\*M\_PI/180.0));

Dy3 = tmpx3\***sin**(((tmpy3+angle)\*M\_PI/180.0));

cout << right;

cout << fixed << setprecision(2) << "\nC0(" << setw(6) << Cx0 << ", " << setw(6) << Cy0 << ") --> D0(" << setw(6) << Dx0 << ", " << setw(6) <<Dy0 << ")";

cout << "\nC1(" << setw(6) << Cx1 << ", " << setw(6) << Cy1 << ") --> D1(" << setw(6) << Dx1 << ", " << setw(6) << Dy1 << ")";

cout << "\nC2(" << setw(6) << Cx2 << ", " << setw(6) << Cy2 << ") --> D2(" << setw(6) << Dx2 << ", " << setw(6) << Dy2 << ")";

cout << "\nC3(" << setw(6) << Cx3 << ", " << setw(6) << Cy3 << ") --> D3(" << setw(6) << Dx3 << ", " << setw(6) << Dy3 << ")";

**return** 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Problems Encountered:**

There were only two issues encountered while working on this program. The first issue was displaying the original coordinates. When the program converted the rectangular coordinates to polar coordinates, the original idea was to rewrite the former rectangular coordinates variables in order to save memory space. However, that made it impossible to report the original coordinates, so a series of temporary variables (tmp) were created to hold the values being converted. The second issue was an order of operations problem. Though the polar to rectangular coordinate conversion formula was correctly applies, adjusting the angle was problematic. The first two coordinates were rotated smoothly, but the program failed to follow the order of operations for the last two coordinates, thus outputting incorrect coordinates. The issue was fixed when parentheses were applied to the formula to simplify the order of operations.

**Testing:**

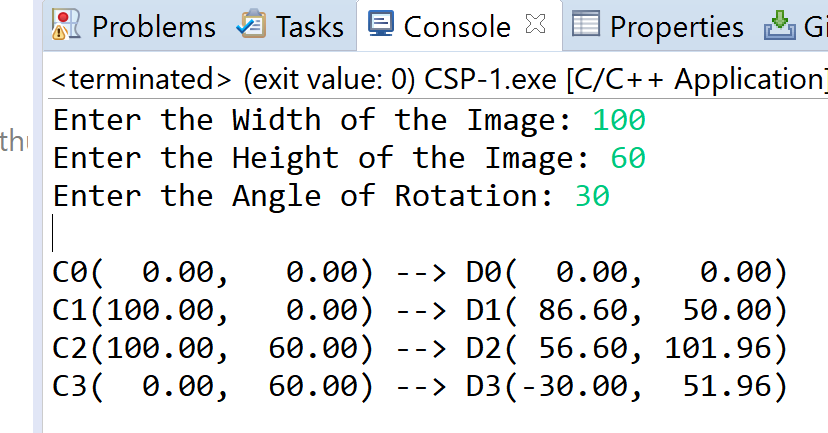
Using the example problems given in the instructions, the example test data was inputted into the code, then compared to the example answers to ensure accuracy.

An example test set of data is: Width of Image: 100

Height of Image: 60

Angle of Rotation: 30

This example was to prove that the program could produce accurately calculated coordinates.

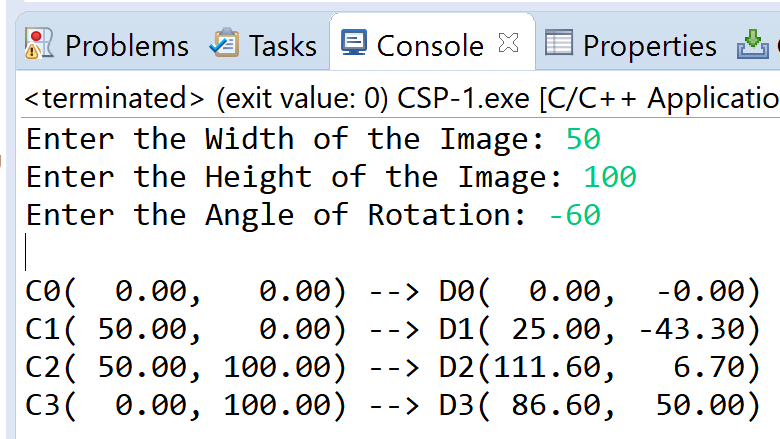


Another example test set of data is: Width of Image: 50

Height of Image: 100

Angle of Rotation: -60

This specific example was to prove that the program could handle negative inputs and still produce accurate results.



**Discussion:**

No issues were found between the program and test data, apart from the issues fixed during debugging. The program was successfully written.