/\* COMPRO1 Machine Problem Part 3 \*/

/\* CHUA\_KYLE MATTHEW C. \*/

/\* S19A \*/

/\* \*/

/\* This file contains the implementation of the mp3\_robot functions \*/

/\* \*/

/\* NOTES: \*/

/\* 1. Change the word "lastname" in the filename to your own last name. \*/

/\* For example, if your last name is SANTOS, then this file should be \*/

/\* named as santos\_mp3\_robot.c \*/

/\* 2. Your main task is to fill-up the body of the functions. \*/

/\* \*/

/\* 3. The following are NOT ALLOWED: \*/

/\* a. calling printf() inside the functions except in DisplayStatus(), \*/

/\* and Quit() functions. \*/

/\* b. addition of new codes other than the implementation of the body \*/

/\* of the functions and the entries in this preamble \*/

/\* c. changing the return type, function name and parametrizations \*/

#include <stdio.h>

#include "mp2\_math.h"

#include "mp3\_robot.h"

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

Initialize/Reset Command

Task:

Initialize/ReSet the robot’s position to (0, 0), and its orientation to 0 degree.

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

void InitializeReset(float \*pfRobotX, float \*pfRobotY, double \*pdRobotAngle)

{

\*pfRobotX=0;

\*pfRobotY=0;

\*pdRobotAngle=0;

}

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

Display Status Command

Task:

Display the robot current state, i.e., its current position

and orientation (in degrees). For the robot position, there should be

four digits after the decimal point. For the orientation, there should

be 2 digits after the decimal point. Output should always start on a

new line.

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

void DisplayStatus(float fRobotX, float fRobotY, double dRobotAngle)

{

printf("\n(%.4f,%.4f) %.2lf degrees\n", fRobotX, fRobotY, dRobotAngle);

}

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

Forward Translation Command

Pre-Condition:

a. fDistance contains the translation distance

b. \*pfRobotX is the robot's current x-coordinate (before translation)

c. \*pfRobotY is the robot's current y-coordinate (before translation)

d. dRobotAngle is the robot's current orientation

Post-Condition:

a. \*pfRobotX is the robot's new x-coordinate (after translation)

b. \*pfRobotY is the robot's new y-coordinate (after translation)

Task:

Translate the robot forward by a distance equivalent to fDistance.

Compute the robot’s new coordinates.

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

void TranslateForward(float fDistance, float \*pfRobotX, float \*pfRobotY, double dRobotAngle)

{

\*pfRobotX += (fDistance\*cosine(dRobotAngle\*PI/180));

\*pfRobotY += (fDistance\*sine(dRobotAngle\*PI/180));

}

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

Backward Translation Command

Pre-Condition:

a. fDistance contains the translation distance

b. \*pfRobotX is the robot's current x-coordinate (before translation)

c. \*pfRobotY is the robot's current y-coordinate (before translation)

d. dRobotAngle is the robot's current orientation

Post-Condition:

a. \*pfRobotX is the robot's new x-coordinate (after translation)

b. \*pfRobotY is the robot's new y-coordinate (after translation)

Task:

Translate the robot forward by a distance equivalent to fDistance.

Compute the robot’s new coordinates.

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

void TranslateBackward(float fDistance, float \*pfRobotX, float \*pfRobotY, double dRobotAngle)

{

\*pfRobotX -= (fDistance\*cosine(dRobotAngle\*PI/180));

\*pfRobotY -= (fDistance\*sine(dRobotAngle\*PI/180));

}

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

Counterclockwise Rotation Command

Pre-Condition:

a. dTheta is the angle of rotation

b. \*pdRobotAngle is the robot's current orientation (before rotation)

Post-Condition:

a. \*pdRobotAngle is the robot's new orientation (after rotation)

Task:

Rotate the robot counterclockwise by an angle equivalent to dTheta.

Compute the robot’s new orientation.

VERY IMPORTANT NOTE!!! Make sure that you keep the value of

\*pdRobotAngle within the range -360 to 360 degrees.

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

void RotateCounterClockwise(double dTheta, double \*pdRobotAngle)

{

while(dTheta>360)

dTheta -= 360;

while(dTheta<-360)

dTheta += 360;

\*pdRobotAngle += dTheta;

}

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

Clockwise Rotation Command

Pre-Condition:

a. fTheta is the angle of rotation

b. \*pdRobotAngle is the robot's current orientation (before rotation)

Post-Condition:

a. \*pdRobotAngle is the robot's new orientation (after rotation)

Task:

Rotate the robot clockwise by an angle equivalent to dTheta.

Compute the robot’s new orientation.

VERY IMPORTANT NOTE!!! Make sure that you keep the value of

\*pdRobotAngle within the range -360 to 360 degrees.

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

void RotateClockwise(double dTheta, double \*pdRobotAngle)

{

while(dTheta>360)

dTheta -= 360;

while(dTheta<-360)

dTheta += 360;

\*pdRobotAngle -= dTheta;

}

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

Quit Simulation Command

Task:

Simply display a message “END SIMULATION”.

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

void Quit(void)

{

printf("\nEND SIMULATION\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF THIS FILE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/