DS4 EQUALIZER

LAB 6 SECTION 1

SUBMITTED BY:

ETHAN GRIESMAN

SUBMISSION DATE:

10/16/2022

Problem:

The objective for this lab was to print out a graph/wave that represents the pitch and or roll of the Dualshock 4 controller based on the button the user pressed. The lab wanted us to print out I's or r's based on which direction the controller was oriented. It then had us print 0's in the center of the screen if the controller was not moving.

Analysis:

The problem states that we have to use actual characters to represent amounts of data and use those to create a repeating graphical representation of the DualShock data. So my program needs to take in the DualShock data, convert it to a useable form, analyze how many characters should be displayed on screen from the converted data, and then display them.

Design

Most of the design for this lab was already supplied to us from the beginning Our mission then was to properly implement the skeleton given to us. To start, I defined an additional 3 variables to be utilized in my main. One was for the time to be scanned and the other two were to be used as my state variables to either be printing roll or pitch values. The next thing in my main was to setup an infinite while loop. This first step was to call the read_line function. After that, I calculated the radian values of the roll and pitch by utilizing the roll and print functions. These were made to have our values easier to work with. They return a proper radian value for both the roll and pitch by using the arcsine function. Next, I used the scaleRadsForScreen function on both roll and pitch so they would print properly. This function used the print_line function in order to accomplish our goal. Graph_line passed the calculated number and its corresponding number out to print_line. Print_line then used the number to see when to start and stop printing out the provided letter. The final step was to adjust the while to stop when the left button was pressed, which I accomplished by setting the condition to stop when the button value was one. All in all, the program had the following design:

1) Read in the data from the Dualshock controller.
2) Convert data to a useful format
3) Analyze how many characters need to be printed for an accurate graphical representation.
4) Print that data

RESULT:
Demo'd to TA 10/12/2022

Testing

Running the program, I was sure to be continuously checking to make sure that the I's. r's, and 0's printed as they were supposed to. Also checking to make sure the buttons worked as intended and switched states, and then killed the program as well.

Comments

While my code and demo went very well, with very little errors, I definitely struggled to grasp pointers at first. Hopefully with increased use, I'll become more and more familiarized with them.

Question #1:

I scaled my values as follows: scale = rad * (78/PI); because the terminal is around 80 spaces wide by default and it looks concise when printed in that format.

Question #2:

Each letter of the graph represents 4.5 degrees as there are a possible 80 spaces to occur so 360/80 = 4.5. As you push to the extremes of the graph it tends to flatten out and jump to its ends, so to speak

```
// 185 lab6.c
// Ethan Griesman Lab section 6
// 10/16/2022
// This is the outline for your program
// Please implement the functions given by the prototypes below and
// complete the main function to make the program complete.
// You must implement the functions which are prototyped below exactly
//  as they are requested.

#include <stdio.h>
#include <math.h>
#define PI 3.141592653589

//NO GLOBAL VARIABLES ALLOWED


//PRE: Arguments must point to double variables or int variables as appropriate
//This function scans a line of DS4 data, and returns
//  True when the square button is pressed
//  False Otherwise
//This function is the ONLY place scanf is allowed to be used
//POST: it modifies its arguments to return values read from the input line.
int read_line(double* g_x, double* g_y, double* g_z, int* time, int* Button_T, int* Button_X, int*
Button_S, int* Button_C);

// PRE: -1.0 <= x_mag <= 1.0
// This function computes the roll of the DS4 in radians
// if x_mag outside of -1 to 1, treat it as if it were -1 or 1
// POST: -PI/2 <= return value <= PI/2
double roll(double x_mag);

// PRE: -1.0 <= y_mag <= 1.0
// This function computes the pitch of the DS4 in radians
// if y_mag outside of -1 to 1, treat it as if it were -1 or 1
// POST: -PI/2 <= return value <= PI/2
double pitch(double y_mag);


// PRE: -PI/2 <= rad <= PI/2
// This function scales the roll value to fit on the screen
// POST: -39 <= return value <= 39
int scaleRadsForScreen(double rad);

// PRE: num >= 0
// This function prints the character use to the screen num times
// This function is the ONLY place printf is allowed to be used
// POST: nothing is returned, but use has been printed num times
void print_chars(int num, char use);

//PRE: -39 <= number <=39
// Uses print_chars to graph a number from -39 to 39 on the screen.
// You may assume that the screen is 80 characters wide.
```

```
void graph_line(int number);



int main()
{
	double x, y, z;			// magnitude values of x, y, and z
	int b_Triangle, b_X, b_Square, b_Circle;	// defines int values for controller buttons
	int tPressed = 1;
assigns a status of 1 to triangle button
	int xPressed = 0;
assigns a status of 0 to x button
	char use;

	double roll_rad, pitch_rad;		// value of the roll and pitch measured in radians
	int scaled_value, time, numRads;		// value of the roll adjusted to fit screen display

	do
	{
		read_line(&x, &y, &z, &time, &b_Triangle, &b_X, &b_Square, &b_Circle);
Get line of input


	if(b_Triangle == 1){ // calculate roll and pitch.  Use the buttons to set the condition for roll
and pitch
		tPressed = 1;
		xPressed = 0;
	}
	else if(b_Square == 1){
		tPressed = 0;
		xPressed = 1;
	}

roll_rad = roll(x);
pitch_rad = pitch(y);

	if(tPressed == 1){
		numRads = scaleRadsForScreen(roll_rad);
	}
	else if(xPressed == 1){
		numRads = scaleRadsForScreen(pitch_rad);
	}


	graph_line(numRads);

		fflush(stdout);
	} while (b_Square != 1); // Modify to stop when the square button is pressed

	return 0;
}
```

```
//define functions

int read_line(double* g_x, double* g_y, double* g_z, int* time, int* Button_T, int* Button_X, int*
Button_S, int* Button_C){ //
	scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d", time, g_x, g_y, g_z, Button_T,
Button_X, Button_S, Button_C);
	if(*Button_S){
		return 1;
	}
	return 0;
}

int scaleRadsForScreen(double rad){
	return (rad * (39/(PI/2)));
}

double roll(double x_mag){ //restrains the values of roll(x) to within a range of -1 and 1
	if(x_mag <= -1){
		x_mag = -1;
	} else if (x_mag >= 1){
		x_mag = 1;
	}
		return asin(x_mag);
}

double pitch(double y_mag){ //restrains the values of pitch(y) to within a range of -1 and 1
	if(y_mag <= -1){
		y_mag = -1;
	} else if (y_mag >= 1){
		y_mag = 1;
	}
		return asin(y_mag);

}

void print_chars(int num, char use){
	for (int i = 0; i < num; i++){
		printf("%c", use);
	}
}

void graph_line(int number){
	if (number > 0){
		print_chars(39 - number, ' ');
		print_chars(number, 'l');
	}
	if (number < 0){
		print_chars(39, ' ');
		print_chars(fabs(number), 'r');
	}
	if (number == 0){
		print_chars(39, ' ');
```

```
                print_chars(1, '0');
        }
        print_chars(1, '\n');
}
```