# Lab Report 08

**LAB #8**

**SECTION #5**

**Westin Gjervold**

**SUBMISSION DATE:**

**11/08/2022**

**10/25/2022**

## Problem

Part 1: Create a moving average function in lab8_1.c. A moving average of length n computes the average of the last n inputs. For instance, a moving average of length 2 of using (1, 3, 5, 6, 3) as the input is (2, 4, 5.5, 4.5). A moving average of length 3 of the same data would be (3, 4.666, 4.666). Also use the moving average function that you just created to move the avatar left and right in lab8_2.c. Also create an argument in lab8_2.c that waits some number of milliseconds and then moves the avatar down the screen.

Part 2: Finish the maze in the lab8_2.c file by adding obstacles by randomly generating a "maze" which the avatar must navigate through to get to the bottom of the screen and ending the program if there are no more available moves.

## Analysis

Part 1: In the moving average (m_avg) function that has the parameters of a buffer array, average size integer, and a new item to add to the array. Then there are two loops, the first one moves all the values in buffer one space to the left and adds the new value to the array. The second loop then adds all the values in the array then divides it by the average size and returns that value. Then in the lab8_2.c file, the program uses the moving average and calculate roll functions to move the avatar either left or right. Also, in the lab8_2.c file, I created a loop that forces the program to wait a designated amount of time (1 second) and then moves the avatar down.

Part 2: In the next part of this lab, I created two functions that randomly assigns boundaries to the maze and then draws those boundaries on the screen. In the first function (generate_maze) uses the difficulty level given by the user and generates more walls if that number is higher. In the generate_maze function there is a loop inside of a loop that either assigns each (x, y) value of the screen with either a wall character or an empty space character. The next function (draw_maze) then two loops again to print all the (x, y) value of the screen with their respective values. Then in the main function I wrote a variety of if statements that force the program to end if there are no more available moves.

## Design

```c
/*-------------------------------------------------------------------------
-                           SE 185 Lab 08
-            Developed for 185-Rursch by T.Tran and K.Wang
-   Name: Westin Gjervold
-   Section: 5
-   NetID: Westing
-   Date: 10/25/2022
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
-                               Includes
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>

/*-------------------------------------------------------------------------
-                               Defines
-------------------------------------------------------------------------*/
#define MAXPOINTS 10000

/*-------------------------------------------------------------------------
-                               Prototypes
-------------------------------------------------------------------------*/
/* Updates the buffer with the new_item and returns the computed
moving average of the updated buffer */
double m_avg(double buffer[], int avg_size, double new_item);

/*-------------------------------------------------------------------------
-                               Implementation
-------------------------------------------------------------------------*/
int main(int argc, char* argv[]) {

    /* DO NOT CHANGE THIS PART OF THE CODE */
    double x[MAXPOINTS], y[MAXPOINTS], z[MAXPOINTS];
    double new_x, new_y, new_z;
    double avg_x, avg_y, avg_z;
    int lengthofavg = 0;
    if (argc>1) {
        sscanf(argv[1], "%d", &lengthofavg );
        printf("You entered a buffer length of %d\n", lengthofavg);
    }
    else {
        printf("Enter a length on the command line\n");
        return -1;
    }
    if (lengthofavg <1 || lengthofavg >MAXPOINTS) {
        printf("Invalid length\n");
        return -1;
    }
```

```c
    for(int i = 0; i < lengthofavg; i++)
    {
        scanf("%lf, %lf, %lf", &new_x, &new_y, &new_z);
        x[i] = new_x;
        y[i] = new_y;
        z[i] = new_z;
    }

    while(1)
    {
        scanf("%lf, %lf, %lf", &new_x, &new_y, &new_z);

        avg_x = m_avg(x, lengthofavg, new_x);
        avg_y = m_avg(y, lengthofavg, new_y);
        avg_z = m_avg(z, lengthofavg, new_z);

        printf("RAW, %lf, %lf, %lf, AVG ,%lf, %lf, %lf\n", new_x, new_y, new_z, avg_x, avg_y, avg_z);
        fflush(stdout);
    }

}

double m_avg(double buffer[], int avg_size, double new_item)
{
    double bufferTotal = 0;
    double bufferAvg = 0;
    for (int i = 0; i < (avg_size - 1); i++){
        buffer[i] = buffer[i+1];
    }
    buffer[avg_size - 1] = new_item;
    for (int i = 0; i < avg_size; i++){
        bufferTotal = bufferTotal + buffer[i];
    }
    bufferAvg = (bufferTotal / avg_size);
    return bufferAvg;
}
```

```c
/*-----------------------------------------------------------------------------
-                           SE 185 Lab 08
-              Developed for 185-Rursch by T.Tran and K.Wang
-   Name: Westin Gjervold
-   Section: 5
-   NetID: Westing
-   Date: 10/25/2022
-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
-                                Includes
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncurses/ncurses.h>
#include <unistd.h>
#include <time.h>

/*-----------------------------------------------------------------------------
-                                Defines
-----------------------------------------------------------------------------*/
#define PI 3.14159
#define COLUMNS 100
#define ROWS 20
#define AVATAR 'A'
#define WALL '*'
#define EMPTY_SPACE ' '
#define NUM_SAMPLES 10

/*-----------------------------------------------------------------------------
-                                Static Data
-----------------------------------------------------------------------------*/
char MAZE[COLUMNS][ROWS];

/*-----------------------------------------------------------------------------
-                                Prototypes
-----------------------------------------------------------------------------*/
void generate_maze(int difficulty);

void draw_maze(void);

void draw_character(int x, int y, char use);

int calc_roll(double mag);

double m_avg(double buffer[], int avg_size, double new_item);

```

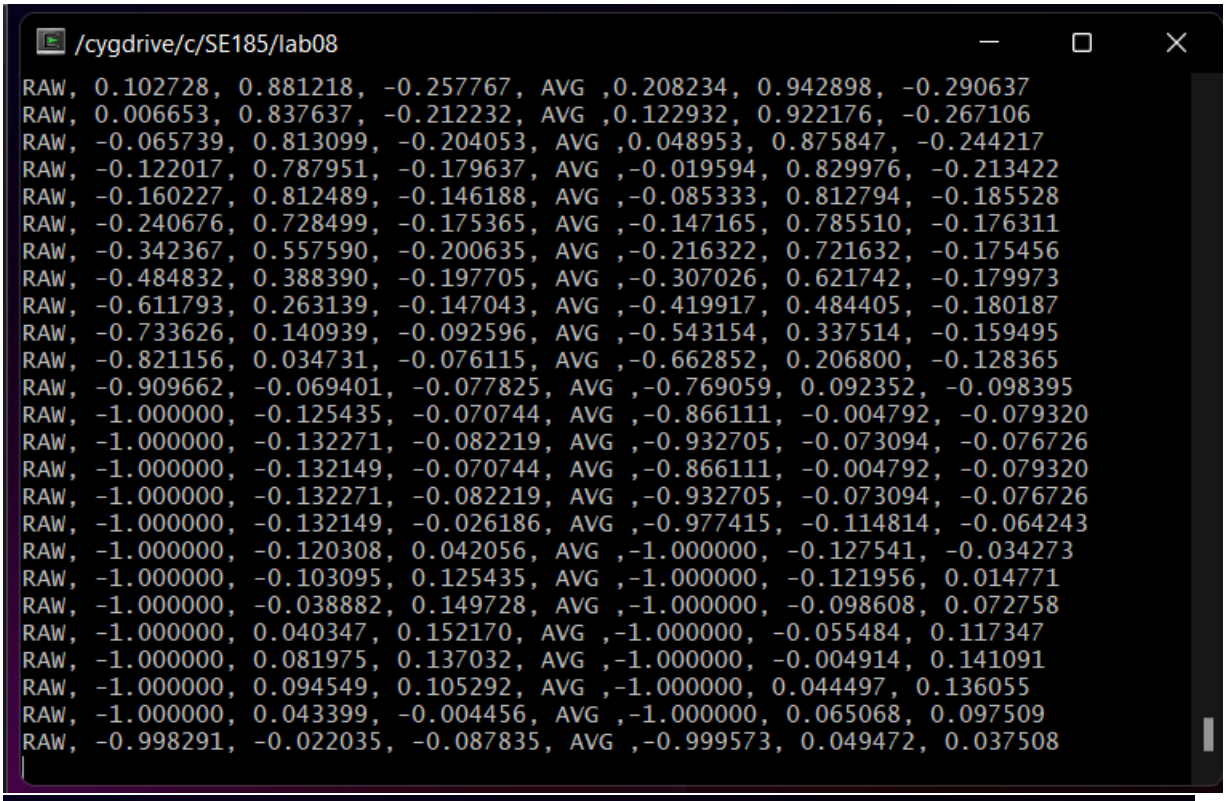
```
/*-------------------------------------------------------------------------
-                              Implementation
-------------------------------------------------------------------------*/
/*  Main - Run with './ds4rd.exe -t -g -b' piped into STDIN */
void main(int argc, char* argv[]){
    //Declaring variables
    int old_X_Avatar = 50;
    int new_X_Avatar = 50;
    int Y_Avatar = 0;
    int difficulty = 0;
    double Xg, Yg, Zg;
    int t, waitTime;
    double x_mag;
    int avg_size = 10;
    double buffer[avg_size];
    int lose = 0;
    sscanf(argv[1], "%d", &difficulty);
    srand((unsigned) difficulty);
    // Setup screen
    initscr();
    refresh();
    // Generate and draw the maze, with initial avatar
    generate_maze(difficulty);
    draw_maze();
    draw_character((COLUMNS/2), 0, AVATAR);
    // Read gyroscope data to get ready for using moving averages.
    scanf("%d, %lf, %lf, %lf", &t, &Xg, &Yg, &Zg);
    // Event loop
    do
    {
        //Read data, update average
        scanf("%d, %lf, %lf, %lf", &t, &Xg, &Yg, &Zg);
        //Is it time to move?  if so, then move avatar
        waitTime = t + 1000;
        //Time delay for Avatar to move
        while(t < waitTime) {
            scanf("%d, %lf, %lf, %lf", &t, &Xg, &Yg, &Zg);
            x_mag = m_avg(buffer, avg_size, Xg);
        }
        //Move right
        if(calc_roll(x_mag) == 1){
            new_X_Avatar = old_X_Avatar + 1;
            if(MAZE[new_X_Avatar][Y_Avatar] == WALL){
                lose = 1;
                break;
            }
            else if(MAZE[new_X_Avatar][Y_Avatar] == EMPTY_SPACE){
                draw_character(new_X_Avatar, Y_Avatar, AVATAR);
                draw_character(old_X_Avatar, Y_Avatar, EMPTY_SPACE);
                old_X_Avatar = new_X_Avatar;
            }
        }
```

```
        //Move left
        else if(calc_roll(x_mag) == -1){
            new_X_Avatar = old_X_Avatar - 1;
            if(MAZE[new_X_Avatar][Y_Avatar] == WALL){
                lose = 1;
                break;
            }
            else if(MAZE[new_X_Avatar][Y_Avatar] == EMPTY_SPACE){
                draw_character(new_X_Avatar, Y_Avatar, AVATAR);
                draw_character(old_X_Avatar, Y_Avatar, EMPTY_SPACE);
                old_X_Avatar = new_X_Avatar;
            }
        }
        //If Avatar is stuck
        else if((MAZE[old_X_Avatar - 1][Y_Avatar] == WALL) && (MAZE[old_X_Avatar + 1][Y_Avatar] == WALL) && (MAZE[new_X_Avatar][Y_Avatar + 1] == WALL)){
            lose = 1;
            break;
        }
        //If Avatar exceeds boundaries
        if((new_X_Avatar < 0) || (new_X_Avatar > COLUMNS)){
            lose = 1;
            break;
        }
        //Move down
        if(MAZE[new_X_Avatar][Y_Avatar + 1] != WALL){
            Y_Avatar++;
            draw_character(new_X_Avatar, Y_Avatar, AVATAR);
            draw_character(new_X_Avatar, Y_Avatar - 1, EMPTY_SPACE);
        }
    } while(Y_Avatar < ROWS); // Change this to end game at right time
    //End window and print Win or Lose message
    endwin();
    if (lose == 1){
        printf("YOU LOSE!\n");
    }
    else{
        printf("YOU WIN!\n");
    }
}

//Calculates and returns moving average
double m_avg(double buffer[], int avg_size, double new_item){
    double bufferTotal = 0;
    double bufferAvg = 0;
    for (int i = 0; i < (avg_size - 1); i++){
        buffer[i] = buffer[i+1];
    }
    buffer[avg_size - 1] = new_item;
    for (int i = 0; i < avg_size; i++){
        bufferTotal = bufferTotal + buffer[i];
    }
    bufferAvg = (bufferTotal / avg_size);
    return bufferAvg;
}
```

```
156  //Draws characters
157  void draw_character(int x, int y, char use){
158      mvaddch(y,x,use);
159      refresh();
160  }
161
162  //Generates maze
163  void generate_maze(int difficulty) {
164      int maze_difficulty;
165      srand(time(NULL));
166      for(int column = 0; column < COLUMNS; ++column){
167          for(int row = 0; row < ROWS; ++row){
168              maze_difficulty = rand() %100;
169              maze_difficulty -= difficulty;
170              if (maze_difficulty <= difficulty){
171                  MAZE[column][row] = WALL;
172              }
173              else{
174                  MAZE[column][row] = EMPTY_SPACE;
175              }
176          }
177      }
178  }
179
180  //Draws maze
181  void draw_maze(void){
182      for(int column = 0; column < COLUMNS; ++column) {
183          for(int row = 0; row < ROWS; ++row) {
184              if (MAZE[column][row] == WALL) {
185                  draw_character(column, row, WALL);
186              }
187              else if(MAZE[column][row] == EMPTY_SPACE){
188                  draw_character(column, row, EMPTY_SPACE);
189              }
190          }
191      }
192  }
193
194  //Calculates and returns roll
195  int calc_roll(double mag){
196      int move;
197      if (mag > 0.7){
198          move = (-1);
199      }
200      else if (mag < -0.7){
201          move = 1;
202      }
203      else{
204          move = 0;
205      }
206      return move;
207  }
208
```

## Testing


/cygdrive/c/SE185/lab08
RAW, 0.102728, 0.881218, -0.257767, AVG ,0.208234, 0.942898, -0.290637
RAW, 0.006653, 0.837637, -0.212232, AVG ,0.122932, 0.922176, -0.267106
RAW, -0.065739, 0.813099, -0.204053, AVG ,0.048953, 0.875847, -0.244217
RAW, -0.122017, 0.787951, -0.179637, AVG ,-0.019594, 0.829976, -0.213422
RAW, -0.160227, 0.812489, -0.146188, AVG ,-0.085333, 0.812794, -0.185528
RAW, -0.240676, 0.728499, -0.175365, AVG ,-0.147165, 0.785510, -0.176311
RAW, -0.342367, 0.557590, -0.200635, AVG ,-0.216322, 0.721632, -0.175456
RAW, -0.484832, 0.388390, -0.197705, AVG ,-0.307026, 0.621742, -0.179973
RAW, -0.611793, 0.263139, -0.147043, AVG ,-0.419917, 0.484405, -0.180187
RAW, -0.733626, 0.140939, -0.092596, AVG ,-0.543154, 0.337514, -0.159495
RAW, -0.821156, 0.034731, -0.076115, AVG ,-0.662852, 0.206800, -0.128365
RAW, -0.909662, -0.069401, -0.077825, AVG ,-0.769059, 0.092352, -0.098395
RAW, -1.000000, -0.125435, -0.070744, AVG ,-0.866111, -0.004792, -0.079320
RAW, -1.000000, -0.132271, -0.082219, AVG ,-0.932705, -0.073094, -0.076726
RAW, -1.000000, -0.132149, -0.070744, AVG ,-0.866111, -0.004792, -0.079320
RAW, -1.000000, -0.132271, -0.082219, AVG ,-0.932705, -0.073094, -0.076726
RAW, -1.000000, -0.132149, -0.026186, AVG ,-0.977415, -0.114814, -0.064243
RAW, -1.000000, -0.120308, 0.042056, AVG ,-1.000000, -0.127541, -0.034273
RAW, -1.000000, -0.103095, 0.125435, AVG ,-1.000000, -0.121956, 0.014771
RAW, -1.000000, -0.038882, 0.149728, AVG ,-1.000000, -0.098608, 0.072758
RAW, -1.000000, 0.040347, 0.152170, AVG ,-1.000000, -0.055484, 0.117347
RAW, -1.000000, 0.081975, 0.137032, AVG ,-1.000000, -0.004914, 0.141091
RAW, -1.000000, 0.094549, 0.105292, AVG ,-1.000000, 0.044497, 0.136055
RAW, -1.000000, 0.043399, -0.004456, AVG ,-1.000000, 0.065068, 0.097509
RAW, -0.998291, -0.022035, -0.087835, AVG ,-0.999573, 0.049472, 0.037508


/cygdrive/c/SE185/lab08