



אוניברסיטת בן-גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת מכונות



פרויקט בתכנות

מספר קבוצה: 23

אלעד ניומן: 302727508

יעקב לבנה : 308372879

עומר טל : 302224019

אורי שוורץ : 021918388

תאריך הגשה : 2/7/2017

שם המדריך : שי שרעבי

-----//FINAL PROJECT//-----

elad newman: 302727508

yaakov livne: 308372879

omer tal: 302224019

ori shwartz: 021918388

/*

//the list of the libraries that the code need to include for using it's functions

#include <GL/glut.h <

#include <stdio.h <

#include <stdlib.h <

#include <math.h<

#include <string.h<

//struactions definition

typedef struct list_3closest { //the struct defines a station that will use in the
program to built a linked list of GAAGs/L.G.O.D (large granes of dust{

int name; //that varoable hold the
"name"/index of the station/GAAG (possible name is an integer that is equal to 1 and above)

int x; //that variable holds the x
coordinate of the center of the GAAG

int y; //that variable holds the y
coordinate of the center of the GAAG

struct list_3closest *first; //definition of pointer which points on another
station/GAAG (which is one of the 3 closest GAAGs)

struct list_3closest *second; //definition of pointer which points on
another station/GAAG (which is one of the 3 closest GAAGs)

struct list_3closest *third; //definition of pointer which points on another
station/GAAG (which is one of the 3 closest GAAGs)

```
double shortpath; //that variable holds a number which is help to sign a
route to the first GAAG which the GAAG includes in (the number is the sum of a distances between GAAGs in the
route)
```

```
{list_closest;
```

```
typedef struct //struct point to use for the GL
```

```
}
```

```
float x; //x point coordinates
```

```
float y; //y point coordinates
```

```
{point;
```

```
typedef struct
```

```
}
```

```
int width, height, offset;
```

```
{INFOHEADER; //structure used in function "read_Info" to extract information from the BMP's header.
```

```
typedef struct
```

```
}
```

```
int power;
```

```
int flag;
```

```
{pix; //structure first used in Matrix_Creation function. the initial pixel's data in matrix
```

```
typedef struct Dust
```

```
}
```

```
int size;
```

```
int xmin;
```

```
int xmax;
```

```
int ymax;
```

```
int ymin;
```

```

        double sumPower;

{Dust; //strucute first used in "DD" function for the first pixel of the future-checked GAG

typedef struct final
}

        int x;

        int y;

        int size;

        double avgPower;

        struct final* next;

{final; //structure first used in "DD" function. containing the final and necessary values and composing the linked
list.

typedef struct
}

        char string;[80]

        int size;

{infoGAG;          // a structure that is used for "section_2" function to copy lines from the text into, and
saving the information of the gag size written in it.

int gagToRemove = 0;                                //global int for display functions

                                                    //functions prototypes

int is_BMP(FILE* picf);

                                                    //verifies correct file
format(type,bits number)

INFOHEADER read_Info(FILE* picf);

                                                    //extracting important info from file's header

pix** Matrix_Creation(FILE* picf, int width, int height, int offset);
        //creating a matrix of pixels structure (struct pix)

```

```

void DD(pix** pixel, int height, int width);
//DD="Dust Detector". using the pixels matrix. finding "GAAGS",
getting its sizes, coordinates, average power(of each one) and the total amount of GAAGS into structures and link it
in a list. all by using internal functions. starts scanning matrix for GAAGS.

void Summerize(pix** pixel, int height, int width, int i, int j, Dust* dustp);
//scanning for relevant "neighbors" in recursion, marks them, and sums intensity of the GAAG candidate
and its size.

final* Add_To_Tail(final* head, Dust* NewData);
//puts temp struct to its final struct and links it in a list ( GAAGS list)

void Make_Text(final* head);
//opens a text file, writing a data from a
given linked list

void section_10;
//apply the section 1 functions for
the program

void section_20;
//The function uses the text file that
was created in section 1, to print for the user a list of the 5 largest GAGs

int input_control(int lowerLimit, int upperLimit);
//function which check if the input is correct/copitable to the program

int searchar(FILE *filepointer, char charforsrch);
//char search in a text file- the function looks for a specific char in a text
and change the pointer pointing address to the address of the next char

int num_of_lines(char *filename);
//number of lines counting in a text file

double distance(int firstPoint[], int secendPoint[]);
//calculate the distance between 2 points

char centers_mat_define(int ***cmatrix, char *filename, int *rows);
//matrix of GAAGs/L.G.O.D (large grains of dust) data definer

double short_route(list_closest* tree, double *dist, double *shortdist);
//function which sign the short route from a chosen GAAG to the first GAAG

void print_route(list_closest* tree, double shortdist, int i, int width);
//function which prints on the screen the shortest route from the chosen GAAG to the first GAAG

```

```

int closest(int** cmatrix, int index, int size, int alreadylinked1, int alreadylinked2);           //find
the closest GAAG      -a function which finds which of the GAAGs in the matrix is the very close to a specific
GAAG

list_closest** list_definer(int** cmatrix, int num_of_objects);
                        //linked list of L.G.O.D/GAAG definer (large granes of dust)

list_closest* build_tree(list_closest* map, list_closest* tree);
                        //function which build a tree out of a direcional map

void delete_tree(list_closest* head);
                        //the function deletes the given tree

void free_pointers(list_closest** map, int **matrix, int rows_num);
                        //function which frees the memory  which was allocated during section_3 functions action

void section_3();
                        //section 3-manu-> instructions for
user+producing a map and a short route

char pic_copy(const char* pic_address, const char* copy_adress);
                        //function which makes a copy of a bmp file

char draw_line(int x1, int y1, int x2, int y2);
                        //function which draw a red line in a bmp file between two points

char draw_red_route(list_closest* tree, double shortdist);
                        //the function draw the shortest route in red colour

char draw_plus(int x, int y);
                        //function which draw "green plus" in
dustcopy.bmp from a received coordinates

char draw_green_coordinates(list_closest** gaagList, int numOfgaag, double shortdist);
                        //the function draws green pluses on all the centers of the GAAGs

void display();
                        //build the display of the
GL

void section_4(int argc, char** argv);
                        // build the graph

float * all_distance(int ** maxrix, float * allsize);
                        //calculate the distance between all the gags

void Prints_a_number_of_points(point allPoint[], int pixels, float* color, int numberOfPoints);      // Prints
lines between a number of different points

```

```
void Prints_two_dots(point allPoint[], int pixels, float* color, int numberOfPoints);  
    // Prints lines between two different points
```

```
void drawText(const char *message, point point, float* color);  
    //send txt message to print on the GL window
```

```
    //Waiting for the user to take action on a keyboard or mouse and exit if the user presses a key (koby kivne)
```

```
Coordinate X (int) Coordinate Y (int) of the position of the action and the key the user clicked on (unsigned char) //input-
```

```
- none //output
```

```
void myKey(unsigned char key, int x, int y) { exit ; (0) }
```

```
int* dustiest_sqr(int edge);  
    //This function locates the corner of the brightness  
square in the image
```

```
void colored_square(int x1, int y1, int size, int choice, int intensity);  
    //The function creates a new BMP file that shows the dustiest square according to the user's color and  
brightness choice
```

```
void section5();  
    //This function prints the menu and  
calls the needed functions for sections 5
```

```
main made by all of us( //the
```

```
main code----- //the
```

```
-----//  
-----
```

```

void main(int argc, char** argv)
{
    int userInput = 0;
    while (userInput != 9)
    {
        printf_s("-----\nNano mechine services\n-----\nMenu: \n1. Scan
dust.\n2. Top 5.\n3. Route it.\n4. Energy cosumption report.\n5. Students addition\n9. Exit.\nEnter choice: ");

        userInput = input_control;(9 ,1)

        switch (userInput)
        {
            case 1:
                section_1;()

                break;

            case 2:
                section_2;()

                break;

            case 3:
                section_3;()

                break;

            case 4:
                section_4(argc, argv);

                break;

            case 5:
                section5;()

                break;

            case 9:
                puts("\nGood bye!");

                exit; (0)
        }
    }
}

```



```

        break;

    default:

        printf_s("\nBad input, try again\n");

        break;

    {

        puts; (""")

    {

{

//functions setting-----
-----
-----
-----

//section_ALL- general functions for different sections use setting

//function which check if the input is correct/copitable to the program (elad newman)

//input- the lower and the upper input's acceptable limits

//output- the user's input/'0' in case of an inappropriate value

int input_control(int lowerLimit, int upperLimit)

}

    int manuInput = 0; //uses for receive the users input

    int check = 0; //holds "1" when an "error" accrues

    char error = '\n'; //receives each character accrues

    do

    ///do-while" loop which act as long as the types keys without pressing enter

    }

    scanf_s("%c", &error, 1); //gets
one character from the user

```

```

        if (error == '\n')    break;
//when the users print enter the loop stops

        if ((manuInput == 0) && (error == '0'))        check = 1;
//if the users typed only '0' sign "error" by insert '1' to check variable

        else

        }

        if ((error < 58) && (error > 47))        manuInput = manuInput * 10 + (error % 48);
//if the character is a digit inserts it to manuInput. use multiplation by 10 for locate the digit it's correct
place

        else        check = 1;

                                                                    //if the input is
not a digit sign "error" by insert '1' to check variable

        {

        { while (error != '\n');                                                                    //when the user
press enter- break the loop

        if ((manuInput <= upperLimit) && (manuInput >= lowerLimit) && (!check)) return manuInput;
//if the input was correct and in the right range- return the input

        else

        return 0;

                                                                    //else- sign "incorrect input" by
returning '0'

        {

//char search in a text file- the function looks for a specific char in a text and change the pointer pointing address to
the address of the next char        (elad newman)

//input- text file pointer and a char which the function will try to find

//output- '0' if it failed with finding the char/'1' if the char has been found

int searchchar(FILE *filepointer, char charforsrch)

}

        int charcomper;
//will use for holding the current char

        while ((charcomper = fgetc(filepointer)) != EOF)        //runs the loop until the end of file
is arrived (increase pointer and make it point on the next char)

        }

```

```

        if (charcomper == charforsrch) return 1;                                //if the current
char is equal to the char which the function looks for, returns 1 and quits

    {

        return 0;
        //returns 0 if the end of the file is arrived without finding an equal char

    }

//number of lines counting in a text file      (elad newman)

//input- the text file name/address

//output- the number of the lines in the file/'-1' if it failed with open the file in the given address

int num_of_lines(char *filename)

{

    int linescounter = 0;                                                        //will use for counting and
holding the amount of lines

    char currentchar;                                                            //will use for
holding the current char which located in the current address

    FILE *tempointer;
    //uses temp pointer for not chage the original pointer of the file

    fopen_s(&tempointer, filename, "rt");                                        //open the txt file
in the address for reading

    if (!tempointer)    return -1;                                              //returns -1 in case of an
address error

    while ((currentchar = fgetc(tempointer)) != EOF)                            //runs the loop until the end of file is arrived
(increase pointer and make it point on the next char)

    {

        if (currentchar == '\n') linescounter++;                                //increases the lines counter when
crossing '\n' value

    {

        fclose(tempointer);                                                    //close the file/release the pointer
from pointing at the file

        return linescounter;                                                    //returns the
result of the counting

    {

```

//matrix of GAAGs/L.G.O.D (large grains of dust) data definer(elad newman)

//input- an int*** pointer which is point on the int** matrix pointer which define in the main, the text file name/address and a pointer of the variable which will hold the matrix number of lines in the main program

//output- the function build a matrix which every row include the relevant GAAGs data. the function returns 'v' when it succeed with build the matrix,and 'f','d' or '0' for reporting an error

char centers_mat_define(int ***cmatrix, char *filename, int *rows)

}

FILE *tempointer;

//temporary file

pointer which the function uses for "move" between characters in the file

fopen_s(&tempointer, filename, "rt");

//opens the file for reading and sets

tempointer as the pointer of the file

if (!tempointer) return 'f';

//returns 'f' if there is

problem with the file/file address

int linesnum = (num_of_lines(filename) - 2), i;

//linesum holds the

number of the lines of the necessary data in the file (use "-2" because the first 2 lines doesn't include necessary data), i uses as index for the loop

int endoftext;

//helps

to detect the end of the file and to detect a problem when the end of the text comes before all the matrix defined

char skipchar;

//helps to skip

irrelevant characters

if (linesnum < 1)

//checks if there is not enough relevant lines or the

num_of_lines function failed

}

fclose(tempointer);

//close the file/release the

pointer from pointing at the file

return '0';

//returns

'0' if there is no relevant information in the file/possible address error or wrong text's structor format

{

*rows = linesnum;

//put the

number of the rows in the futured matrix in the content of the "rows" ponting address

*cmatrix = (int**)malloc(sizeof(int)*(linesnum));

//defines the rows in the matrix (linesnum-2

because the dust file first 2 lines doesn't include necessary data)

if (*cmatrix == NULL)

//checks

if the dynamic set succeeded

```

    }

    fclose(tempointer); //close the
file/release the pointer from pointing at the file

    return 'd'; //returns
'd' when failed with define the matrix

    {

        for (i = 0; i < (linesnum); i++) //a loop for defining the cells of the rows and fill them
with the compatible data

    }

    (*cmatrix)[i] = (int*)malloc(sizeof(int) * 3); //defines current row cells. define 2 cells- for
x and y coordinates and another one which will hold the size of the GAAG and will use to define the linked list in
section_3

    if ((*cmatrix)[i] == NULL) //checks if the dynamic set
succeeded

    }

    fclose(tempointer); //close the
file/release the pointer from pointing at the file

    return 'd'; //returns 'd' when
failed with define the matrix

    {

        endoftext = searchchar(tempointer, '('); //looking for the next')

        if (!endoftext)

    }

    fclose(tempointer); //close
the file/release the pointer from pointing at the file

    return 'b'; //returns 'b' if the
text is finished before the matrix is fully defined or the text format doesn't fits to this program

    {

        fscanf_s(tempointer, "%d%c%d", &(*cmatrix)[i][0], &skipchar, 1, &(*cmatrix)[i][1]);
//reads the x and y values from the file and insert them in the first two row's cells. use "skipchar" for
skipping the ',' character by receiving it to that variable

```

```

        searchchar(tempointer, '\t');
                                                    //skips chars until finding tab-> promote file pointer
to the character that come right after tab

        fscanf_s(tempointer, "%d", &(*cmatrix)[i][2]);
        //receives the size value and insert it to the third cell

        searchchar(tempointer, '\n');
                                                    //use searchchar to jump to the next line in the file

    {

        fclose(tempointer);
                                                    //close the file/release the pointer
from pointing at the file

        return 'v';
                                                    //reports that the definition succeeded by
returning 'v'

    {

//calculate the distance between 2 points (koby livne)

//input- 2 int pointers ,the first cell contains the X coordinate of the point and the second cell contains the Y
coordinate of the point

//output - distance the 2 pointers that you send to the function (double)

double distance(int * firstPoint, int * secendPoint)

    {

        double x = pow(firstPoint[0] - secendPoint[0], 2);
                                                    // calculate the coordinate x for the sqrt

        double y = pow(*(++firstPoint) - *(++secendPoint), 2); // calculate the coordinate y for the sqrt

        return sqrt(x + y);
                                                    //
        return the distance the 2 pointers that you send to the function

    {

//section_1 functions setting

void section_1()

    {

        pix** matrix;

        int height, width, offset, i;

        INFOHEADER info;

```

```

FILE* picf;

fopen_s(&picf, "d2.bmp", "rb"); //opens the file at reading binaric mode

if (!picf) //checks if file is successfully opened
    printf_s("\nProblem open the file d2.bmp\n");
else
}

if (is_BMP(picf)) //checks correct format and bits number
}

    info = read_Info(picf); //extracting info fro, header

    height = info.height; //aplying the info which was just extrated into variables

    width = info.width;

    offset = info.offset;

    matrix = Matrix_Creation(picf, width, height, offset); //creates pixels matrix

    DD(matrix, height, width); //main functions (checkelaborations)


    for (i = 0; i < height; i++)
    }

        free(matrix[i]);    //memory release

    {

        free(matrix);

    }

    {

        fclose(picf);

    }

//Omer Tal

//verifies correct file format(type,bits number)

//input: file adress

```

//output: exits with the code 0 if not, returning 1 if yes

```
int is_BMP(FILE* picf)
```

```
    char type[3];    //creating a string to compare
```

```
    unsigned short int bits;
```

```
    fseek(picf, 0, 0);
```

```
    fread(type, 1, 2, picf); //putting in type the 2 letters picture type from header
```

```
    fseek(picf, 28, 0);
```

```
    fread(&bits, 1, 2, picf); //putting in bits number of bits in each pixel from header
```

```
    if (!strcmp(type, "BM") || (bits != 24)) {    //checking existence of 2 must conditions to 24bits bmp file. if its  
not, exit. if yes continue
```

```
        printf("\nThis d2.bmp format is not supported - convert file to 24bppRBG\n");
```

```
        return 0;
```

```
    {
```

```
        return 1;
```

```
    {
```

```
//Omer Tal
```

```
//extracting important info from file's header
```

```
//input: file adress
```

```
//output: filling struct INFOHEADER info, turning adress
```

```
INFOHEADER read_Info(FILE* picf)
```

```
    INFOHEADER info;
```

```
    //offset
```

```
    fseek(picf, 10, 0);
```

```
    fread(&info.offset, 1, 4, picf);
```



```

        //Image Width in pixels

        fseek(picf, 18, 0);

        fread(&info.width, 1, 4, picf);


        //Image Height in pixels

        fseek(picf, 22, 0);

        fread(&info.height, 1, 4, picf);


        return(info);
    }


//Omer Tal

//creating a matrix of pixels structure (struct pix)

//input: file adress, dimensions, offset(when to start reading)

//output: pointer to a new made pixels struct matrix


pix** Matrix_Creation(FILE* picf, int width, int height, int offset)
{
    int i, j, padding;


    pix **pixel;


    pixel = (pix**)malloc(sizeof(pix*)*height); //creating a pixel matrix

    for (i = 0; i < height; i++)

    {
        pixel[i] = (pix*)malloc(sizeof(pix)*width);

        {
            for (i = 0; i < height; i++) //checking allocation

            }
        }
    }
}

```

```

        if (!pixel[i])
            exit(1)
    }

    padding = 4 - (width * 3) % 4; //getting number of bytes to pad in each row
    if (padding == 4)
        padding = 0;
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
            fseek(picf, offset + ((i*width + j) * 3) + i*padding, SEEK_SET); //filling the matrix in
            picture's pixels. actual insertion of Red color only, ignoring GB(not relevant)

            pixel[i][j].power = 0; //preparing the cell

            fread(&pixel[i][j].power, 1, 1, picf); //including the padding for a correct
            filling (the edit to the fseek makes the reader to jump from SET to the correct place

            pixel[i][j].flag = 0; //preparing the cell

        }

        for (j = width * 3; j % 4 != 0; j++) //skipping the padding
        {
            fgetc(picf);
        }
    }

    return (pixel); //returning the pixle matrix (adress) for its future use
}

//Omer Tal

```

//DD="Dust Detector". using the pixels matrix. finding "GAAGS", getting its sizes, coordinates, average power(of each one) and the total amount of GAAGS into structures and link it in a list. all by using internal functions. starts scanning matrix for GAAGS.

//input: pixel's matrix adress and its dimensions

//output: none

void DD(pix** pixel, int height, int width)

{

int i, j, size, X, Y;

double AvgPower;

Dust dust; //multi use structure (recycled)

Dust* dustp = &dust;

final* head;

int count = 0;

head = NULL;

for (i = 0; i < height; i++) //scanning the pixel's matrix

{

for (j = 0; j < width; j++)

{

if (pixel[i][j].flag == 0 && pixel[i][j].power >= 80) //checking if checked yet(flag=0) and in sufficient power for composing GAAG

{

dustp->sumPower = 0; //preparing the structure's fields

dustp->size = 0;

dustp->xmax = j;

dustp->xmin = j;

dustp->ymax = i;

dustp->ymin = i;

Summerize(pixel, height, width, i, j, dustp); //filling fields by internal function(size, xmin,xmax,ymin,ymax, sumpower). marking checked pixels

```

        if (dustp->size >= 30) //checking if the current struct is GAAG
        }

        count++; //GAAG's counter

        if (count == 1) //first gaag unique case- has to be printed and it's Head of
list
        }

        AvgPower = (dustp->sumPower / dustp->size); //calculating
final values from the current Dust structure

        X = (((dustp->xmax) + (dustp->xmin)) / 2) + 1;
        Y = (((dustp->ymin) + (dustp->ymin)) / 2) + 1;
        size = dustp->size;

        printf_s("\nCoordinate (%d,%d)\nSize %d\nAverage intensity
%.1lf\n", X, Y, size, AvgPower);

        head = malloc(sizeof(Dust)); //allocating memory

        head->avgPower = AvgPower; //filling values from the
current Dust structure(current GAAG) in its final structure

        head->x = X;
        head->y = Y;
        head->size = size;
        head->next = NULL; // preparing list

        {
        else
        }

        Add_To_Tail(head, dustp); //adding new GAAGs to head one
after one creating a list

        {

        {

```

```

        {

        {

printf_s("Total of %d big dust particles.\n", count);

Make_Text(head);

{

//Omer Tal

//scanning for relevant "neighbors" in recursion, marks them, and sums intensity of the GAAG candidate and its size.

//input: pixel's matrix pointer, matrix dimensions, pixels location (first time is the bottom left in the GAAG), temp
struct pointer to store values

//output: none

void Summerize(pixel** pixel, int height, int width, int i, int j, Dust* dustp)

}

    pixel[i][j].flag = 1; //mark the scanned pixel for not being checked again

    if (i < height - 1 && pixel[i + 1][j].flag == 0 && pixel[i + 1][j].power >= 80) //checking exceeding from matrix
limits, checking 4 optional pixel "neighbors" being in sufficient power and not marked

    //                                } if neighbors are verified- do same action on them

        Summerize(pixel, height, width, i + 1, j, dustp);

    {

    if (i > 0 && pixel[i - 1][j].flag == 0 && pixel[i - 1][j].power >= 80)

    }

        Summerize(pixel, height, width, i - 1, j, dustp);

    {

    if (j < width - 1 && pixel[i][j + 1].flag == 0 && pixel[i][j + 1].power >= 80)

    }

        Summerize(pixel, height, width, i, j + 1, dustp);

    {

    if (j > 0 && pixel[i][j - 1].flag == 0 && pixel[i][j - 1].power >= 80)

```

```

    }

    Summerize(pixel, height, width, i, j - 1, dustp);

    {

    dustp->size++; //sums number of pixels in the GAAG

    dustp->sumPower += pixel[i][j].power; //sums total power of pixels in the GAAG

    if (i > dustp->ymax)

        dustp->ymax = i; //finding the extrimest coordinates of the GAAG (no need to check down)

    if (j > dustp->xmax)

        dustp->xmax = j;

    if (j < dustp->xmin)

        dustp->xmin = j;

    {

//Omer Tal

//puts temp struct to its final struct and links it in a list ( GAAGS list)

//input: list's head adress, temp struct

//output: list's head adress

final* Add_To_Tail(final* head, Dust* NewData)

}

    final* newfinal = (final*)malloc(sizeof(final)); //allocating memory

    newfinal->avgPower = NewData->sumPower / NewData->size; ////filling values from the current Dust
    structure(current GAAG) in its final structure

    newfinal->x = (NewData->xmax + NewData->xmin) / 2 + 1; //adding plus 1 to round up

    newfinal->y = (NewData->ymax + NewData->ymin) / 2 + 1;

    newfinal->size = NewData->size;

    newfinal->next = NULL; //new struct pointing null

    final* temp = head; //temp pointer

    while (temp->next != NULL) //pointer moving to last in list

```

```

    }

    temp = temp->next;

    {

temp->next = newfinal; //getting the data last in list.

return head;

{

//Omer Tal

//opens a text file, writing a data from a given linked list

//input: list's head address

//output: none

void Make_Text(final* head)

}

    final* temp;

    FILE* dust;

    fopen_s(&dust, "dust.txt", "w"); //opens file at writing only mode

    if (dust == NULL) //checking if the file is successfully opened

    }

        printf_s("failed to open dust.txt");

        return; //gets out of the function if not successfully opened.

    {

temp = head;

    fputs("coordinate\tsize\taverage Intensity\n===== \t===== \t===== \n", dust);
//writing title

    while (temp != NULL) //going from head till last in list and writing values as below.

    }

        fprintf(dust, "(%d,%d) \t%d \t%.1f\n", temp->x, temp->y, temp->size, temp->avgPower);

        temp = temp->next; //moving to next in list each loop

```

```

{

while (head != NULL) //memory release
}

temp = head;

head = head->next; //going from first till last, each time saving first one, moving to next and delete
the saved one

free(temp);

{

fclose(dust);

{

//section_2 function prints for the user the sizes of the 5 largest GAGAs (Ori Swarcztz)

//The function used the TXT file that was created in section 1, and gets the information that is in it.

//Every line of the text is copied in an infoGAG struct array. that contains a string and an int.

//The function find in the text the size of the GAG and saves it into the int and then sort the array by GAG size before
printing the first 7 lines.

//input - nothing.

//output - nothing or exits with the code 0

void section_2} 0

char * dust = "dust.txt"; //creating a char array with the name of the text file

char tempChar;

int i, j, flag = 0;

int GAGsize;

int tempSize = 0;

long numOfLines = num_of_lines(dust); //reciving into numOfLine the number of of lines in
the text using 'num_of_lines' function

infoGAG *infoMatrix = (infoGAG*)malloc(sizeof(infoGAG)*numOfLines); //creating an
array of infoGAG stucture .

```



```

//in this structure there is an string to copy the line of text into

//and a int that will contain the size of the GAG that is written in
that line

infoGAG tempInfo; //creating a temporary
infoGAG sturact variable to be used for sorting the GAGs by size

FILE * txt; //opening the text
file

fopen_s(&txt, dust, "rt");

if (!txt) (exit(0)); //if the file does not open the program exits
with code 0

for (i = 0; i < numOfLines; i++) { //A loop runs for every line in the text and copy if to a
the appropriate string in the infoMatrix array

    for (j = 0; j < 79; j++) //The text file was created using this
program and the line length is known to be less that 79 characters long.

    }

    tempChar = fgetc(txt); //a char is from the text is saved
into tempChar

    if (tempChar == '\n') //if the char is '\n' then the loop got
to the end of the line and

    } //in that
case 0 is added to the string to symbolize the end of the line

    infoMatrix[i].string[j] = 0; //and the loop stops coping the line.

    break;

    {

        infoMatrix[i].string[j] = tempChar; //in case the end of the line was not reached, the char
that was saved in tempchar is added to the string

        {

```

```

{

    infoMatrix[1].size = 0;                //the 2 first line in the text does not contain the information of
GAGS, so the GAG size is zeroed

    infoMatrix[2].size = 0;

    for (i = 2; i < numOfLines; i++)    //for the other lines the information of the GAG size is saved as an int in
the array
    {

        for (j = 0; j < 79; j++)        //the function runs along the characters of the line
        {

            if (infoMatrix[i].string[j] == 9)    //in the text 2 TABs separate the size of the
GAG from the rest of the line from each side.

                }

            //TAB ascci number is 9. when the first TAB is found a while loop starts .

            //the while loop continues till the next TAB is found.

            while (infoMatrix[i].string[j + 1] != 9)

                }

            tempSize = tempSize * 10;                //in every repetition of the
loop the next digit of a number is found .

            //in order to add it correctly to the last know size of the GAG is multiplied by 10

            GAGsize = (int)infoMatrix[i].string[j + 1] - 48 + tempSize;
            //The sum of the last known size is added to the new digit that was found in the text

            //The digit was saved as a char in the text, in
order to get the correct value -

            48//is subtracted form it. because numbers
start at the 48th place in the ascii table/

            tempSize = GAGsize;                //the
GAGsize is saved also in a temporary variable.

```

```

        j++;
// 1 is added to the j, so the next while loop, the next char will be reached in the string.

        flag = 1; //flag is
changed to 1, to notify that the size of the GAG was copied for this line.

        {
            {
                if (flag == 1) //if flag is one
            }

            tempSize = 0; //Tempsize is
zeroed

            infoMatrix[i].size = GAGsize; //the GAG size is saved in it's place
in the array

            flag = 0; //flag is zeroed
for the next loop

            break; //the
line loop ends

            {
                {

                    for (i = 2; i < numOfLines; i++) //The outer loop sort the array by the size of the
GAGs, starting in the 3rd line and stopping in the end

                    } //every
size is being compared to the size that comes after it, and if it is smaller, they change place

                    for (j = 2; j < numOfLines - 1; j++) { //this loop runs from the start from the 3rd line to the
line before the end .

                        if (infoMatrix[j].size < infoMatrix[j + 1].size) //if the GAG size is smaller that then one that
comes after it

                        }

                        tempInfo = infoMatrix[j]; //the infoGAG stract of
that line is saved temporary

```

```

        infoMatrix[j] = infoMatrix[j + 1];           //the next infoGAG of the array is
copied into the place of the one before it

```

```

        infoMatrix[j + 1] = tempInfo;               //the information that was
saved replaced the information of that infoGAG that was saved.

```

```

    {
        {
            {

                printf_s("\nBiggest dust particles found: \n");           //printing a announcement for the user

                for (i = 0; i < 7; i++)                                     //printing the 7
first stings of the array for the user
            }

            puts(infoMatrix[i].string);

            {

                fclose(txt);                                               //closing
the TXT FILE
            }

```

```

//section_3 functions setting

```

```

//find the closest GAAG    -a function which finds which of the GAAGs in the matrix is the very close to a specific
GAAG    (elad newman)

```

```

//input- the GAAGs matrix, the index of the chosen GAAG, the size- amount of rows of the matrix and the indexes of
two possible GAAG which is already linked to the that GAAG (the chosen GAAG)

```

```

//output- the function returns the index of the GAAG which it finds as most close (considering the terms)

```

```

int closest(int** cmatrix, int index, int size, int alreadylinked1, int alreadylinked2)
{

```

```

    int i, returned_index = -1;           //i will use as the index of the loop, returned_index will hold
the value which the function returns ("-1" sign of "no GAAG matches)

```

```
double dist = INFINITY; //will hold the
closest distance value which is found between the chosen GAAG to the tested GAAG. set as INFINITY to assure that
the next relevant distance found will be with a smaller value
```

```
for (i = 0; i < size; i++)
//the loop is go through each row/GAAG and check if it compatible to considered as "the closest GAAG"
}
```

```
if ((index != i) && (cmatrix[i][2] < 3) && (i != alreadylinked1) && (i != alreadylinked2))
//first checks the terms-the test GAAG is not the chosen one, that GAAG is not already linked to the chosen
GAAG and checks if that GAAG is not poited by 3 others GAAGs
```

```
}
```

```
if (dist > (distance(cmatrix[index], cmatrix[i])))
//checks if the distance between that two GAAGs is smaller than the shortest distance which is found so far
}
```

```
dist = distance(cmatrix[index], cmatrix[i]);
//if it smaller, set it as the shortest distance by insert it's value to "dist"
```

```
returned_index = i;
//set the retuned index as i- means returned_index holds now the index of the closest
GAAG so far
```

```
{
```

```
{
```

```
{
```

```
if (returned_index != -1) cmatrix[returned_index][2]++; //if the function
finds a "closest GAAG" increace the third cell by one. that cell counting how many GAAGs pointing to the GAGG in
that index
```

```
return returned_index;
//returns the value of "retuned_index"- the closest GAAG's index/"-1" in case of no match
```

```
{
```

```
//linked list of L.G.O.D/GAAG definer (large granes of dust) (elad newman)
```

```
//input- the GAAGs matrix and the number of the GAAGs that need to define
```

```
//output- the function return an "array" of lists_closest* (pointers) which every "row" points on a "lists_closest"
pointer
```

```
list_closest** list_definer(int** cmatrix, int num_of_objects)
```

```
}
```

```

list_closest** array_of_lists;

//array_of_lists- each "row" points on a station in the future linked list

array_of_lists = (list_closest**)malloc(num_of_objects * sizeof(list_closest));
//memory allocation for the array. the malloc will allocate enough place for variables from type
"list_closest*" "num_of_objects" times (the number GAAGs which the function receives)

int i, closest1, closest2, closest3;
//i-index for the loops, closest1/2/3 will hold the indexes of the GAAGs in the matrix that
are "busy" (already linked)

for (i = 0; i < num_of_objects; i++)
//loop which runs "num_of_objects" time and define each GAAG station in the future liked list
}

array_of_lists[i] = (list_closest*)malloc(sizeof(list_closest)); //allocates memory for
each row in the size of one station (list_closest type size)

array_of_lists[i]->name = i + 1;
//for each row defines the name of the GAAG (the digits from 1 to num_of_objects)

array_of_lists[i]->x = cmatrix[i][0]; //for each row
defines "x" coordination of the GAAG

array_of_lists[i]->y = cmatrix[i][1];
//for each row defines "y" coordination of the GAAG

array_of_lists[i]->first = NULL; //sets
the "first" pointer as NULL

array_of_lists[i]->second = NULL; //sets
the "second" pointer as NULL

array_of_lists[i]->third = NULL; //sets
the "third" pointer as NULL

array_of_lists[i]->shortpath = INFINITY;
//sets for each row "shortpath" with an INFINITY value

{
for (i = 0; i < num_of_objects; i++)
}

closest1 = closest(cmatrix, i, num_of_objects, -1, -1);
//calls closest function and insert the returned index of the first most closest GAAG to "closest1"

if (closest1 != -1) array_of_lists[i]->first = array_of_lists[closest1];
//checks if the function returned "-1" means there is no available. true- set the "first" pointer

```

```

        closest2 = closest(cmatrix, i, num_of_objects, closest1, -1); //calls
closest function and insert the returned index of the second most closest GAAG to "closest2"

        if (closest2 != -1) array_of_lists[i]->second = array_of_lists[closest2];
//checks if the function returned "-1" means there is no available. true- set the "second" pointer

        closest3 = closest(cmatrix, i, num_of_objects, closest1, closest2); //calls
closest function and insert the returned index of the second most closest GAAG to "closest3". tell the function which
cells is already pointed by this GAAG by send closest1 and closest2 to the it

        if (closest3 != -1) array_of_lists[i]->third = array_of_lists[closest3];
//checks if the function returned "-1" means there is no available. true- set the "third" pointer

    {

        return array_of_lists;
//the function the built array

    }

//function which build a tree out of a direcional map (elad newman)

//input- direcional map and an empty pointer from struct "list_closest"*

//output- the "head" of the built tree

list_closest* build_tree(list_closest* map, list_closest* tree)

{

    int temp = map->name; //holds the
current GAAG's name in a temporary int variable

    list_closest* sub_tree; //define a new station
which the tree will be built of

    *tree = *map; //copy the current GAAG
content

    map->name = -1;
// 0 is a sign which marks the station/GAAG as "already visited."

    if (!map->first || (map->first->name == -1)) //go "first "

        tree->first = NULL; //if that
station was already visited or not exist tree->first= null

    else {

        sub_tree = malloc(sizeof(list_closest)); //memory allocation in the
size of one station

```

```

        tree->first = build_tree(map->first, sub_tree);           //apply the function on
(map->first, sub_tree) and insert the GAAG which will be define while the functions action

    {

        if (!map->second || (map->second->name == -1))             //go "second"

            tree->second = NULL;                                   //if that station
was already visited or not exist tree->first= null

        else {

            sub_tree = malloc(sizeof(list_closest));             //memory allocation in the
size of one station

            tree->second = build_tree(map->second, sub_tree);      //apply the function on (map-
>second, sub_tree) and insert the GAAG which will be define while the functions action

            {

                if (!map->third || (map->third->name == -1))        //go "third "

                    tree->third = NULL;                           //if that
station was already visited or not exist tree->first= null

                else {

                    sub_tree = malloc(sizeof(list_closest));      //memory allocation in the
size of one station

                    tree->third = build_tree(map->third, sub_tree); //apply the function on
(map->second, sub_tree) and insert the GAAG which will be define while the functions action

                    {

                        map->name = temp;                          //put the
GAAG name back in the current GAG-name

                        return tree;                               //return the new defined
GAAG station

                    }

                }

            }

        }

    }

//the function deletes the given tree          (elad newman)

//input- a tree "head"

//output- none

void delete_tree(list_closest* head)

}

```



```

        if (head->first)                                //checks in the current GAAG->first points
on another GAAG
    }

        delete_tree(head->first);                        //then- "go first"- apply the function on
GAAG which head->first pointer points on
    {

        if (head->second)                                //checks in the current GAAG->second
points on another GAAG
    }

        delete_tree(head->second);                      //then- "go second"- apply the
function on GAAG which head->second pointer points on
    {

        if (head->third)                                //checks in the current GAAG->third points
on another GAAG
    }

        delete_tree(head->third);                        //then- "go third"- apply the function on
GAAG which head->third pointer points on
    {

        free(head);                                     //free the current GAAG

        return;                                         //finish the action. no value returns
    }

//function which sign the short route from a chosen GAAG to the first GAAG                (elad newman)

//input- tree head, double type pointer which its content is 0 and another double type pointer which its content is
INFINITY (warning: the

//output- the shortest route's total distance which is signs all the GAAGs that are include in that path

double short_route(list_closest* tree, double *dist, double *shortdist)

{

    list_closest* tempointer;

    double tempdist;

    int xy[2] = { tree->x,tree->y };

    int next_xy[2]

```

```

    tempointer = tree;                                //make tempointer to point
on the address which "tree" points on

    tempdist = *dist;                                //make tempdist holds the content
of "dist"

    if (tree->name == 1)                                //checks if the
current GAAG is the GAAG number one

    }

    if (*dist < *shortdist)
//if the current sum of distances is smaller than shortdist do:

    }

    tree->shortpath = *dist;                            //sign
this GAAG as one of the short routes station

    *shortdist = *dist;
//insert the new short distance which found to the content of shortdist pointer

    return *shortdist;
//return the new short distance

    {

    else

        return INFINITY;
//if the routes ends (GAAG one arrived) and this route is not shorter than all the others- return INFINITY

    {

        if (tree = tempointer->first)                    //checks if "first" exists and inserts the next
GAAG address to tree

        }

        next_xy[0] = tree->x;                            //insert the next GAAG x
coordinate to next_xy[0]

        next_xy[1] = tree->y;                            //insert
the next GAAG y coordinate to next_xy[1]

        *dist = tempdist + distance(xy, next_xy);        //add the distance
between the current GAAG to the next one to the sum of the routes distances

        if (*shortdist == short_route(tree, dist, shortdist)) //go "first"- apply the
function on the next "first" GAAG. checks if the function returns the value of shortdist

```

```

tempointer->shortpath = *shortdist;
//then- it means that a new short route have found - sing that this GAAG is belongs to the short
route by inserting shortdist to tempointer->shortpath

{

    if (tree = tempointer->second)                                //checks if "second" exists and
    inserts the next GAAG address to tree

}

    next_xy[0] = tree->x;                                          //inserst the next GAAG x
coordinate to next_xy[0]

    next_xy[1] = tree->y;                                          //inserst
the next GAAG y coordinate to next_xy[1]

    *dist = tempdist + distance(xy, next_xy);                    //add the distance
between the current GAAG to the next one to the sum of the routes distances

    if (*shortdist == short_route(tree, dist, shortdist))        //go "second"- apply the
function on the next "second" GAAG. checks if the function returns the value of shortdist

        tempointer->shortpath = *shortdist;
        //then- it means that a new short route have found - sing that this GAAG is belongs to the short
route by inserting shortdist to tempointer->shortpath

    {

        if (tree = tempointer->third)                            //checks if "third" exists and
        inserts the next GAAG address to tree

    }

        next_xy[0] = tree->x;                                      //inserst the next GAAG x
coordinate to next_xy[0]

        next_xy[1] = tree->y;                                      //inserst
the next GAAG y coordinate to next_xy[1]

        *dist = tempdist + distance(xy, next_xy);                //add the distance
between the current GAAG to the next one to the sum of the routes distances

        if (*shortdist == short_route(tree, dist, shortdist))    //go "third"- apply the
function on the next "third" GAAG. checks if the function returns the value of shortdist

            tempointer->shortpath = *shortdist;
            //then- it means that a new short route have found - sing that this GAAG is belongs to the short
route by inserting shortdist to tempointer->shortpath

    {

```

```

    tree = tempointer;
        //returns the previous/original address to tree- the "current" GAAG address

    if (tempointer->shortpath == *shortdist)    return tempointer->shortpath;    //checks
if this current GAAG belongs to the short route by if checking his shortpath variable equals to shortdist content's
value. then- returns that value

    return INFINITY;
        //in other cases returns INFINITY instead

{

//function which prints on the screen the shortest route from the chosen GAAG to the first GAAG
    (elad newman)

//input- the head of the tree which is actually the chosen GAAG and the sign of the shortest route which is the value
of the sum of the distances between the GAAGs in that route

//output- non

void print_route(list_closest* tree, double shortdist)

}

    if (tree->shortpath == shortdist) printf_s("(%d,%d)", tree->x, tree->y);    //print
the current GAAG's x,y in the right format (if it belongs to the shortest route)

    }

    if (tree->name != 1)    //checks if that
GAAG is not the last station in the route- not GAAG number one. only than continues with the process

    }

    if (tree->first)
        //if tree->first exist then:

    }

    if (tree->first->shortpath == shortdist)    //if tree-
>first->shortpath equals to the sign- shortdist then:

    }

    printf_s("->");

    //print"<->"

    print_route(tree->first, shortdist);    //go "first"- apply
the function on tree->first

    {

```

```

        {
            if (tree->second)
                //if tree->second exist then:
        }

        if (tree->second->shortpath == shortdist) //if tree-
>second->shortpath equals to the sign- shortdist then:
        }

        printf_s("->");
        //print"<-""

        print_route(tree->second, shortdist); //go
"second"- apply the function on tree->second
        {
            {
                if (tree->third)
                    //if tree->third exist then:
            }

            if (tree->third->shortpath == shortdist) //if tree-
>third->shortpath equals to the sign- shortdist then:
            }

            printf_s("->");
            //print"<-""

            print_route(tree->third, shortdist); //go "third"-
apply the function on tree->third
            {
                {
                    {
                        {
                            //function which makes a copy of a bmp file (ori shwartz)

                            //input- the function receives the address/name of the original picture and the addrees/name of the copy that will be
                            create

                            //output- the function returns a char '0'/'1' to report if the process succeed or failed

```

```

char pic_copy(const char* pic_address, const char* copy_address)
}

    FILE * original_pic;                                //definition of
pointer for the original file

    fopen_s(&original_pic, pic_address, "rb");            //open for reading

    if (!original_pic) return 0;                          //return '0' when
the definition fails

    if (is_BMP(original_pic))                            //calling the
functions "is_BMP" to check if the file a BMP-24bit file

    }

    FILE * copied_pic;
//definition of pointer for the copy file

    fopen_s(&copied_pic, copy_address, "wb");            //open for writing

    if (!copied_pic) return 0;
//return '0' when the definition fails

    unsigned int get_char;                                //integer
which will use for receiving one byte from the original file and write it to the copy file

    fseek(original_pic,0,SEEK_SET);
//definition of pointer for the copy file; (

    while ((get_char = fgetc(original_pic)) != EOF) {     //as long as the
current byte is not EOF- the end of the file, do:

        fputc(get_char, copied_pic);
//insert the received byte to the copy. (fputc and fgetc increase the pointer by them selfs)

        {

            fclose(copied_pic);
//close the copy file

            fclose(original_pic);                        //close
the original file

```

```

        return 1; //return
1 which reports that the copy has been created properly

    {

        fclose(original_pic); //close
the original file

        return -1; //if the
picture is not an 24 bit BMP return -1

    {

//function which draw a red line in a bmp file between two points (eladnewman)

//input- the function receives x and y coordinate of two points

//output- the function returns a char '0'/'1' to report if the process succeed or failed

char draw_line(int x1, int y1, int x2, int y2)

    }

    FILE* copyPointer;
    //definition of pointer for the copy file

    fopen_s(&copyPointer, "dustcopy.bmp", "r+b"); //open for read and write
without erase the original file

    if (!copyPointer) return 0;
    //return '0' when the definition fails

    int tempX, tempY, i, j, gap, offset, width, padding; //temporary variables which hold x
and y values, i and j indexes for the loops,gap holds the gap between two coordinates,offset holds the offset of the
bmp file value, width holds the amount of pixels in one row of the picture and padding used for holding the padding
value

    double m = ((double)(y1 - y2) / (double)(x1 - x2)); //m holds the
slope of a the straight line between the two points

    double b = (y1 - (m*x1)); //b holds
it's y intercept

    char fullByte = 255, zero = 0; //fullbyte is one byte
which holds it's maximum value- 255 (all his bits are 1), zero is one byte which icludes only 0 bits

    fseek(copyPointer, 10, SEEK_SET); //move the pointer to the
place which the offset value is located

```

```

        fread(&offset, 4, 1, copyPointer);
variable                                     //insert offset's value to the offset

        fseek(copyPointer, 18, SEEK_SET);
place which the width value is located      //move the pointer to the

        fread(&width, 4, 1, copyPointer);
variable                                   //insert width's value to the width

        padding = width % 4;
insert it to the padding variable          //calculate the padding value and

        if (x1>x2)
point is bigger than the second point- switch between them    //if the x coordinate of the first

        }

        tempX = x2;
                                           //use tempX for holding x2 value

        tempY = y2;
                                           //use tempy for holding y2 value

        x2 = x1;
                                           //put x1 in x2

        y2 = y1;
                                           //put y1 in y2

        x1 = tempX;
                                           //put the former x2 value in x1

        y1 = tempY;
                                           //put the former y2 value in x2

    {

//case of vertical (m=INF/-INF)- doesn't need to assure that it vertical because it is the last option that left

    if (x1 == x2)
                                           //checks if vertical

    }

        if (y1>y2)
                                           //if y1>y2 switch between

them

        }

        tempY = y2;

        y2 = y1;

        y1 = tempY;

    {

        gap = abs(y1 - y2);
//calculate the gap between the two y coordinates. this value is the amount of pixels which need to be
painted red

```



```

        for (i = 0; i <= gap; i++)
            //loop which runs "gap+1" times and makes the movement in y coordinate. "<=" adds to the (y1-y2) "+1".
            this action is because the amount of pixels that is needed a change is include y1 and also y2

        }

        fseek(copyPointer, ((y1 - 1) * (3 * width + padding) + (x1 - 1) * 3) + offset, SEEK_SET);
        //move (y1-1) rows and then move (x1-1) pixels right

        fwrite(&zero, 1, 1, copyPointer);

        //put byte=0 in B-byte

        fwrite(&zero, 1, 1, copyPointer);

        //put byte=0 in G-byte

        fwrite(&fullByte, 1, 1, copyPointer);

        //put byte=255 in R-byte

        y1++;

        //increase y coordinate

    {

        fclose(copyPointer);

        //close
bmp file

        return(1);

        //report that the process
succeed by returning '1'

    {

        //case of m>=1

        if (m >= 1)

        {

            gap = (x2 - x1);

            //calculate the gap between the two x
coordinates

            for (j = 0; j < gap; j++)

                //loop which runs "gap"
times (the x movement)

            {

                while (((y1 - (m*x1 + b)) <= m) && ((int)(y1 - (m*x1 + b)) >= 0))

                    //while the current point (x1,y1) is fits to the equations of the lines between
y=mx+b to y=mx+(b+m) do: (uses (int) to "round" answers that are very close to zero because of calculates
deviation)

                }

```

```

        fseek(copyPointer, ((y1 - 1)*(3 * width + padding) + ((x1 - 1) * 3) + offset),
SEEK_SET);
        //move (y1-1) rows and then move (x1-1) pixels right

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in G-byte

        fwrite(&fullByte, 1, 1, copyPointer);
        //put byte=255 in R-byte

        y1++;
        //increase y1 (y's axis movement)

    {
        x1++;
        //increase x1 (x's axis movement)

    {
        fseek(copyPointer, ((y2 - 1) * (3 * width + padding) + ((x2 - 1) * 3) + offset), SEEK_SET);
        //reach the last pixel at point (x2-1,y2-1)

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in G-byte

        fwrite(&fullByte, 1, 1, copyPointer);
        //put byte=255 in R-byte

        fclose(copyPointer);
        //close the bmp

file
        return(1);
        //return 1, means that the function succeed with drawing the

line
    {
        //case of (m < 1) && (m>0)

        if ((m < 1) && (m>0))

    }

        gap = (y2 - y1);
        //calculate the gap between the two y

coordinates

```

```

        for (j = 0; j < gap; j++)                                //loop which runs
"gap" times (the y movement)
    }

    while (((y1 - (m*x1 + b)) >= (-1)) && ((int)(y1 - (m*x1 + b)) <= 0))
        //while the current point (x1,y1) is fits to the equations of the lines between y=mx+b to
y=mx+(b-1) do: (uses (int) to "round" answers that are very close to zero because of calculates deviation)
    }

    fseek(copyPointer, ((y1 - 1) * (3 * width + padding) + ((x1 - 1) * 3) + offset),
SEEK_SET);
        //move (y1-1) rows and then move (x1-1) pixels right

    fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

    fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

    fwrite(&fullByte, 1, 1, copyPointer);
        //put byte=255 in R-byte

    x1++;
        //increase x1 (x's axis movement)

    {

    y1++;
        //increase y1 (y's axis movement)

    {

    fseek(copyPointer, ((y2 - 1) * (3 * width + padding) + ((x2 - 1) * 3) + offset), SEEK_SET);
        //reach the last pixel at point (x2-1,y2-1)

    fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

    fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in G-byte

    fwrite(&fullByte, 1, 1, copyPointer);
        //put byte=255 in R-byte

    fclose(copyPointer);
        //close the bmp

file

```

```

        return(1);
//return 1, means that the function succeed with drawing the
line
    {
        //case of m<=-1
        if (m <= -1)
        {
            gap = (x2 - x1);
            //calculate the gap between the two x coordinates
            for (j = 0; j < gap; j++)
                //loop which runs
                "gap" times (the x movement)
            {
                while (((y1 - (m*x1 + b)) >= (m)) && ((int)(y1 - (m*x1 + b)) <= 0))
                    //while the current point (x1,y1) is fits to the equations of the lines between
                    y=mx+b to y=mx+(b+m) do: (uses (int) to "round" answers that are very close to zero because of calculates
                    deviation)
                {
                    fseek(copyPointer, ((y1 - 1) * (3 * width + padding) + ((x1 - 1) * 3) + offset),
                    SEEK_SET);
                    //move (y1-1) rows and then move (x1-1) pixels right
                    fwrite(&zero, 1, 1, copyPointer);
                    //put byte=0 in b-byte
                    fwrite(&zero, 1, 1, copyPointer);
                    //put byte=0 in b-byte
                    fwrite(&fullByte, 1, 1, copyPointer);
                    //put byte=255 in R-byte
                    y1--;
                    //decrease y1 (y's axis movement)
                {
                    x1++;
                    //increase x1 (x's axis movement)
                }
                fseek(copyPointer, ((y2 - 1) * (3 * width + padding) + ((x2 - 1) * 3) + offset), SEEK_SET);
                //reach the last pixel at point (x2-1,y2-1)

```

```

fwrite(&zero, 1, 1, copyPointer);
//put byte=0 in b-byte

fwrite(&zero, 1, 1, copyPointer);
//put byte=0 in G-byte

fwrite(&fullByte, 1, 1, copyPointer);
//put byte=255 in R-byte

fclose(copyPointer);
//close the bmp file

return(1);
//return 1, means that the function succeed with drawing the
line
{
//case of (m > -1) && (m<0)
if ((m > -1) && (m<0))
}

gap = (y1 - y2);
//calculate the gap between the two y coordinates

for (j = 0; j < gap; j++)
//loop which runs
"gap" times (the y movement)

}

while (((y1 - (m*x1 + b)) >= (-1)) && ((int)(y1 - (m*x1 + b)) <= 0))
//while the current point (x1,y1) is fits to the equations of the lines between y=mx+b to
y=mx+(b-1) do: (uses (int) to "round" answers that are very close to zero because of calculates deviation)

}

fseek(copyPointer, ((y1 - 1) * (3 * width + padding) + ((x1 - 1) * 3) + offset),
SEEK_SET);
//move (y1-1) rows and then move (x1-1) pixels right

fwrite(&zero, 1, 1, copyPointer);
//put byte=0 in b-byte

fwrite(&zero, 1, 1, copyPointer);
//put byte=0 in b-byte

fwrite(&fullByte, 1, 1, copyPointer);
//put byte=255 in R-byte

x1++;
//increase x1 (x's axis movement)

```

```

    {
        y1--;
        //decrease y1 (y's axis movement)

    {
        fseek(copyPointer, ((y2 - 1) * (3 * width + padding) + ((x2 - 1) * 3) + offset), SEEK_SET);
        //reach the last pixel at point (x2-1,y2-1)

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in G-byte

        fwrite(&fullByte, 1, 1, copyPointer);
        //put byte=255 in R-byte

        fclose(copyPointer);
        //close the bmp file

        return(1);
        //return 1, means that the function succeed with drawing the

```

line

```

    {
        //case of horizontal (m=0)

        if (y1 == y2)
            //checks if the line is horizontal by compare between the two y coordinates

        }

        gap = (x2 - x1);
        //calculate the gap between the two x coordinates

        for (i = 0; i <= gap; i++)
            //loop which runs "gap+1" times and makes the movement in x coordinate. "<=" adds to the (x1-x2) "+1".
            this action is because the amount of pixels that is needed a change is include x1 and also x2

    }

```

```

        fseek(copyPointer, ((y1 - 1) * (3 * width + padding) + ((x1 - 1) * 3) + offset), SEEK_SET);
        //move (y1-1) rows and then move (x1-1) pixels right

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

        fwrite(&zero, 1, 1, copyPointer);
        //put byte=0 in b-byte

```

```

        fwrite(&fullByte, 1, 1, copyPointer);
                                                    //put byte=255 in R-byte

        x1++;
                                                    //increase x1 (x's axis movement)

    {

        fclose(copyPointer);
//close the bmp file

        return(1);
//return 1, means that the function succeed with drawing the line

    {

        return(0);
                                                    //if non
of the m which is calculate has a compatible value returns 0 for alerting an error

    {

//function which draw "green plus" in dustcopy.bmp from a received coordinates
                                                    (eladnewman)

//input- x and y coordinates of the center of the plus

//output- '0' when the function failed, '1' when it succeed

char draw_plus(int x, int y)

    }

    FILE* copyPointer;
//definition of pointer for the copy file

    fopen_s(&copyPointer, "dustcopy.bmp", "r+b");
                                                    //open for read
and write without erase the original file

    if (!copyPointer) return 0;
                                                    //return '0' when the definition fails

    INFOHEADER picInfo;
//picInfo is a variable from the struct INFOHEADER which hold the header info of the picture file

    picInfo = read_Info(copyPointer);
                                                    //insert
the header info into picInfo

    int tempX, tempY, padding;
                                                    //temporary variables which hold x and y values and padding uses for holding the
padding value

    char fullByte = 255, zero = 0;
                                                    //fullbyte is one
byte which holds it's maximum value- 255 (all his bits are 1), zero is one byte which icludes only 0 bits

```

```

padding = picInfo.width % 4;
//calculate the padding and insert it to the padding variable

for (tempX = x - 2; tempX < x + 3; tempX++)
//loop for which runs 5 time and draw the horizontal line of the plus
}

if ((tempX > 0) && (tempX < picInfo.width))
//check if the x coordinate is in the range of
the picture width
}

fseek(copyPointer, ((y - 1) * (3 * (picInfo.width) + padding) + ((tempX - 1) * 3) +
picInfo.offset), SEEK_SET); //move (y1-1) rows and then move (x1-1)
pixels right

fwrite(&zero, 1, 1, copyPointer);

//write '0' byte to the blue byte

fwrite(&fullByte, 1, 1, copyPointer); //write 255 (full byte) to
green byte

fwrite(&zero, 1, 1, copyPointer);

//write '0' byte to the red byte

{
{
for (tempY = y - 2; tempY < y + 3; tempY++)
//loop for which runs 5 time and draw the vertical line of the plus
}

if ((tempY > 0) && (tempY < picInfo.height))
//check if the y coordinate is in the range of the
picture height
}

fseek(copyPointer, ((tempY - 1) * (3 * (picInfo.width) + padding) + ((x - 1) * 3) +
picInfo.offset), SEEK_SET); //move (y1-1) rows and then move (x1-1) pixels right

```



```

        fwrite(&zero, 1, 1, copyPointer);

//write '0' byte to the blue byte

        fwrite(&fullByte, 1, 1, copyPointer);

//write 255 (full byte) to
green byte

        fwrite(&zero, 1, 1, copyPointer);

//write '0' byte to the red byte

    {

    {

        fclose(copyPointer); //close the picture
file

        return 1; //report of success by
returning '1'

    {

//the function draw the shortest route in red colour (eladnewman)

//input- a tree head and the sign of the short route which is the sum of the distances

//output- the function returns '0' when fails and '1' when succeeded

char draw_red_route(list_closest* tree, double shortdist)

    {

        if (tree->name != 1) //checks if that GAAG is
not the last station in the route- not GAAG number one. only than continues with the process

    {

        if (tree->first) //check if "first"
exist

    {

        if (tree->first->shortpath == shortdist)
//check if first is part of the shortest route

    {

        if (!draw_line(tree->x, tree->y, tree->first->x, tree->first->y)) return(0);
//apply draw line function on the centers of the current GAAG and the next first GAAG. if the
function returns '0' - report failure by returning '0'

```

```

draw_red_route(tree->first, shortdist);
        ""move first"-apply the function on the next first
GAAG
    {
    {
    if (tree->second)                                //check if
"second" exist
    }

    if (tree->second->shortpath == shortdist)
        //check if second is part of the shortest route
    }

    if (!draw_line(tree->x, tree->y, tree->second->x, tree->second->y)) return(0);
    //apply draw line function on the centers of the current GAAG and the next second
GAAG. if the function returns '0' - report failure by returning '0'

    draw_red_route(tree->second, shortdist);
        ""move second"-apply the function on the next first GAAG
    {
    {
    if (tree->third)                                //check if "third"
exist
    }

    if (tree->third->shortpath == shortdist)
        //check if third is part of the shortest route
    }

    if (!draw_line(tree->x, tree->y, tree->third->x, tree->third->y)) return(0);
    //apply draw line function on the centers of the current GAAG and the next third GAAG. if the
function returns '0' - report failure by returning '0'

    draw_red_route(tree->third, shortdist);
        ""move third"-apply the function on the next first
GAAG
    {
    {
    {

```

```

        return(1);
        //report of success by returning '1'

{

//the function draws green pluses on all the centers of the GAAGs (eladnewman)

//input- an array of pointers of GAAGs, the amount of the GAAGs and the sign of the short route

//output- the function returns '0' when fails and '1' when succeeded

char draw_green_coordinates(list_closest** gaagList, int numOfgaag, double shortdist)

}

        int i; //the loop index and the index of the
array

        for (i = 0; i < numOfgaag; i++) //loop
which runs as times as the GAAGs number

        }

        if (!(draw_plus(gaagList[i]->x, gaagList[i]->y))) return 0;
        //draw one green plus on the current GAAG center location. if the function "draw plus" returns '0' - report
failure by returning '0'

        {

        return(1);
        //report of success by returning '1'

{

//function which frees the memory which was allocated during section_3 functions action (elad newman)

//input- the directional map/the array of pointers of the GAAGs, the matrix of the GAAGs centers and the number of
the GAAGs

//output- non

void free_pointers(list_closest** map, int **matrix, int rows_num)

}

        int i; //the index of the loop and for the matrix and
the array

        for (i = 0; i < rows_num; i++) //loop for releasing each row in the matrix

        }

        free(matrix[i]); //free one row of the matrix

```

```

        free(map[i]);                                //free the current GAAG pointer in
the array
    {
        free(matrix);                                //free the matrix pointer
        free(map);                                    //free the array pointer
    }
//section 3-manu-> instructions for user+producing a map and a short route        (elad newman)
//input- none
//output- none
void section_3()
{
    list_closest** directional_map;
    //array of pointer of GAAGs
    list_closest* temporartree;
        //a temporary GAAG pointer which will point on the head of the tree
    int **centersmatrix;
                                //matrix of centers of GAAGs
    int amountOfGAAGs, userInput = 0, i, error;
                                //variables which holds the number of the GAAGs, the user input and
the index of the loop, error checks if the returned value from the functions are reporting an error
    double dist1 = 0, distINF = INFINITY;
                                //two variable which holds distances between two
GAAGs. dist one set as 0 and distINF set as INFINITY
    double *pdist1 = &dist1, *pINF = &distINF;
                                //two pointers which theirs contents equal to the two
distances variables
    char *copyAddress = "dustcopy.bmp", *filename = "dust.txt", *pic_address = "d2.bmp";
        //the pictures addresses

    if ((centers_mat_define(&centersmatrix, filename, &amountOfGAAGs)) == 'v')
        //apply function which define the centers matrix and checks if it succseded
}

```

[illegible]

```

        temporartree = build_tree(directional_map[userInput - 1], temporartree);
                                //build the tree

        dist1 = short_route(temporartree, pdist1, pINF);

        //find the short route and hold it's symbol at dist1

        if (dist1 == INFINITY)

        //checks if the route exist

                                printf_s("no route found from (%d,%d) to (%d,%d)!",
directional_map[userInput - 1]->x, directional_map[userInput - 1]->y, directional_map[0]->x, directional_map[0]->y);
        // if there is no route between the GAAGs print a compatible message

        else

        }

                                puts("Best route found: ");

                                print_route(temporartree, dist1);

        //print the route on the screen

                                puts("\n");

                                if (!(error=pic_copy(pic_address, copyAddress))) {

        //make a copy of the d2.bmp picture/file and check if the copy is exist/in the correct format

                                puts("\nfunction copy_pic has difficulties open d2.bmp or
dustcopy.bmp");                                //if not report an
error

                                getchar; 0

                                exit(1);

                                //exit

                                {

                                if (error != -1)

        //check if the the picture is not in the right format

                                }

```

```

puts("Check for a new copy file dustcopy.bmp");

//report when the copy is ready

if (!draw_red_route(temporartree, dist1))

//draw the red route on the copy and check if the function
succsseded

}

puts("\nfunction draw_marks has difficulties open
dustcopy.bmp or create new dustcopy.bmp image with the required marks"); //if not
report an error

getchar; 0

exit(1);

//exit

{

puts("Done connecting biggest dust particles in dustcopy.bmp");

//reporting the program have drawn the route

if (!draw_green_coordinates(directional_map,
amountOfGAAGs, dist1))

//draw green pluses on the centers
of the GAAGs in the copy and check if the function succsseded

}

puts("\nfunction draw_marks has difficulties open
dustcopy.bmp or create new dustcopy.bmp image with the required marks"); //if not
report an error

getchar; 0

exit(1);

//exit

{

puts("Done marking coordinate of biggest dust particles in
dustcopy.bmp");

//reporting the program have drawn the green pluses

{

```

```

        delete_tree(temporartree);

        //delete the tree- free the pointers of the stations in the tree
        {
            {
                { while (!userInput);
                                                    //repit
on "section_3" applications until the input is correct

                free_pointers(directional_map, centersmatrix, amountOfGAAGs);
                //free the pointers- free the centers matrix and the map (GAAGs pointers array)

            {
{

//section_4 functions setting

//start the Gl and Selects the basic parameters from which the GL window will be built from (koby livne)
//input- maim argumaints
//output - nothing

void section_4(int argc, char** argv)
}

char * txt = "dust.txt";
                                                    //The path of the
file from which we read the gags from

int lines = num_of_lines(txt);

if (lines == -1) puts("\nCan't find dust.txt");

if (lines<3) printf_s("\nThere is no valid content in dust.txt");

else

}

printf_s("\n");

```



```

while(1)
{
    if (lines > 11) {
        printf_s("Enter length of route up to 9: ");
        gagToRemove = input_control(9, 1)
        if (gagToRemove > 0 && gagToRemove < 10)
            break;
    }
    else {
        //Acquires the amount of gag the user wants to remove and inserts them into a
global variable

        printf_s("Enter length of route up to %d: ", lines - 2);
        gagToRemove = input_control(1, lines - 2);
        if (gagToRemove > 0 && gagToRemove < (lines - 1))
            break;
    }
}

glutInit(&argc, argv);
//init for the GL

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //Color
mode for the GL

glutInitWindowSize(640, 480);
//Window size for the GL

glutInitWindowPosition(0, 0);
//Window position (from upper left corner) for the GL

glutCreateWindow("Energy consumption in time"); //Create window
with title

glutDisplayFunc(display);
//Present default display and beld all the pixels in the GL window

```

```

        glutKeyboardFunc(myKey);
//Define what keyboard press does(if the user press any key we exit the program) .

        glutMainLoop();
//Start loop and display the all graphics

    {

{

//bild all the display we going to see on the GL window (koby livne)

//input-nothing

//output - nothing

void display()

}

        float color[] = { 0.0,0.0,0.0 };                                //folor we going to use in the GL start in black

        glClearColor(1, 1, 1, 1);                                        //The color of the window (white)

        glClear(GL_COLOR_BUFFER_BIT);                                    // clear window


        point MainblackLinesGraf[] = { { -0.61, 0.7 }, { -0.61, -0.6; { { 0.6- ,0.7 }, { //hold the points for the
main graph lines

        Prints_a_number_of_points(MainblackLinesGraf, 3, color, 3);
//print the main graph lines in black color


        point topBlackLinesEroow[] = { { -0.64, 0.64 }, { -0.61, 0.7//; { { 0.64 ,0.58- }, { hold the points for the top
graph lines eroow

        Prints_a_number_of_points(topBlackLinesEroow, 3, color, 3);
//print the top graph lines eroow in black color


        point botomBlackLinesEroow[] = { { 0.67, -0.63 }, { 0.7, -0.6//; { { 0.57- ,0.67 }, { hold the points for the top
graph lines eroow

        Prints_a_number_of_points(botomBlackLinesEroow, 3, color, 3);
//print the bottom graph lines eroow in black color

```

```

//hold the points for the time and energy
coordinates

point timeEnergyCoordinates[] = { { 0.6, -0.55 }, { 0.6, -0.6 }, { 0.3, -0.55 }, { 0.3, -0.6 }, { -0.0, -0.55 }, { -0.0, -
0.6 }, { -0.3, -0.55 }, { -0.3, -0.6 },
{ 0.3, 0.56 }, { 0.3, 0.61 }, { 0.0, 0.56 }, { 0.0, 0.61 }, { 0.3, 0.56 }, { 0.3, 0.61 }, { 0.6, 0.56 }, { 0.6, 0.61 },
};

Prints_two_dots(timeEnergyCoordinates, 3, color, 16);
//print the time and energy coordinates in black color

//start the calculation for what we
will show on the graph

float energy = 0, time = 0, tempEnergy = 0, currentTime;

point errorPoint = { 0,0 };
//point position for the error Message

int i, error = 0, rows = 0;

char tmpStr[60];
//for convert float to string using sprintf_s

char * txt = "dust.txt";
//The path of the file from which we read the gags from

int ** gagDataMatrix;

float * allsize = (float*)malloc(sizeof(float)*(gagToRemove * 2 - 1)); //malloc that hold
all the size (gag size and the distance between gag size)

if (allsize == NULL) exit(1);
//Checks if the memory has been allocated properly to the malloc and if not exit
from the program

point * allTimeAllEnergy = (point*)malloc(sizeof(point)*(gagToRemove * 2)); //malloc that hold
all the x coordinates

if (allTimeAllEnergy == NULL) exit(1);
//Checks if the memory has been allocated properly to the malloc and if not exit from the
program

```

```

if ((centers_mat_define(&gagDataMatrix, txt, &rows)) == 'b') {
    //insert the data to the gagDataMatrix from the dust file

    puts("function readDust resulted with file read error. Check that the text file has the correct
format.function readDust resulted with file read error. Check that the text file has the correct format.function
readDust resulted with file read error. Check that the text file has the correct format.");

    exit;(1)

{
    all_distance(gagDataMatrix, allsize);
        //fill the allsize list from the gagDataMatrix cordinasen

    allTimeAllEnergy[0].x = 0;
        //start at time=0

    allTimeAllEnergy[0].y = 0;
        //start at energy=0

                                                                    //loop that calculates all the coordinates of
energy and time that I have to display on the graph

    for (i = 0; i < gagToRemove * 2 - 1; i++)
        //A loop that tells whether I am over gag or not

    }

    for (currentTime = 0; currentTime < ((float)(allsize[i] / 100)); currentTime += 0.001f)        //A loop
that takes the time steps of 0.001 and ends when we enter or exit Gag

    }

    tempEnergy = energy;
        //Preserving the initial energy of each step

                                                                    //tempEnergy = The initial energy, ((i) % 2
== 0) = If we are over GAG will return true if we are not over GAG will return a false, allsize[i] = The size of the gag
that we pass over

    energy += (tempEnergy*((i) % 2 == 0)*allsize[i] * 0.75f + 10)*0.001f;
    //Calculate the final energy of each step

    time += 0.001;

```

```

        if ((energy - tempEnergy) / 0.001 >= 550 && error == 0)
            //Checking whether we have exceeded the permitted energy, that is, whether the energy derivative is greater
            than 550, and also checks that we have not exceeded the past

        }

        error = i;

        //Guard in which Gag we exceeded

        errorPoint.y = energy;
        //Keeps the deviation location in terms of energy

        errorPoint.x = time;
        //Keeps the deviation location in terms of time

    {

    {

        allTimeAllEnergy[i + 1].y = energy;
        //Retain the cordent of the energy to be printed on the graph

        allTimeAllEnergy[i + 1].x = time;
        //Retain the cordent of the time to be printed on the graph

    {

        color[2] = 0.7f;

        //Changes the color to blue

        //A loop that inserts all the points
        on the graph and draws the line between them

        for (i = 0; i < gagToRemove * 2 - 1; i++)

        }

        //Holding the points of the function we are printing to the window

        point pointGraf[] = { { -0.61 + (allTimeAllEnergy[i].x) / (0.9*(ceil((time) * 2) / 2)), -0.6 +
        (allTimeAllEnergy[i].y / (ceil((energy) / 10) * 10 - (ceil((energy) / 10) * 10)*0.17)), }, { -0.61 + (allTimeAllEnergy[i + 1].x)
        / (0.9*(ceil((time) * 2) / 2)), -0.6 + (allTimeAllEnergy[i + 1].y / (ceil((energy) / 10) * 10 - (ceil((energy) / 10) * 10)*0.17)) }
        };

        Prints_two_dots(pointGraf, 2, color, 2); // points the points for the function

    {

        color[2] = 0.0; //Changes the color back to black

```



```

//Construct the numbers and units on the scale on the axes

color[1] = 0.3f; //Changes the color to green


point timeCoordinatesMessage = { 0.695, -0.8 }; //The starting point of the txt messeg (for the
time scale)

drawText("Time [s]", timeCoordinatesMessage, color); //Sends the text message
to a function drawText to print it to the screen


point EnergyCoordinatesMessage = { -0.9, 0.75 }; //The starting
point of the txt messeg (for the Enargy scale)

drawText("Enargy [J]", EnergyCoordinatesMessage, color);
//Sends the text message to a function drawText to print it to the screen


point TotalEnergyCoordinatesMessage = { -0.5, 0.9 };
//The starting point of the txt messeg (for the total energy)

sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "Total enargy consumption : %.3f[J]", energy);
//Inserts the doubel into a statement and then to string so we can send it
to a function drawText

drawText(tmpStr, TotalEnergyCoordinatesMessage, color);
//Sends the text message to a function drawText to print it to the screen


//Holds
the points for the values of the coordinates

point coordinatesValue 0.3,0.72- }, { 0.6,0.72- }, { 0.68- ,0.3- }, { 0.68- ,0.0 }, { 0.68- ,0.3 }, { 0.68- ,0.6 } } = []
; { { 0.3- ,0.72- }, { 0.0,0.72- }, {

//Build values and print them on the time scale With an upward circle and a division according to the
location of the cordenta where the first is the general time and each one is less than a quarter of it

sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.2f", (1.05*(ceil((time) * 2) / 2))); //Inserts
the doubel into a statement and then to string so we can send it to a function drawText

```

```
drawText(tmpStr, coordinatesValue[0], color);
//Sends the text message to a function drawText to print it to
the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.2f", ((1.05*(ceil((time) * 2) / 2))) * 3 / 4); //Inserts the
doubl into a statement and then to string so we can send it to a function drawText
```

```
drawText(tmpStr, coordinatesValue[1], color);
//Sends the text message to a function drawText to
print it to the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.2f", (((1.05*(ceil((time) * 2) / 2)))) / 2); //Inserts the
doubl into a statement and then to string so we can send it to a function drawText
```

```
drawText(tmpStr, coordinatesValue[2], color);
//Sends the text message to a function drawText to print it to
the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.2f", (((1.05*(ceil((time) * 2) / 2)))) / 4); //Inserts the
doubl into a statement and then to string so we can send it to a function drawText
```

```
drawText(tmpStr, coordinatesValue[3], color);
//Sends the text message to a function drawText to print it to
the screen
```

```

//Build
values and print them on the energy scale With an upward circle and a division according to the location of the
cordenta where the first is the general energy and each one is less than a quarter of it
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.0f", ceil((energy) / 10) * 10);
//Inserts the doubl into a statement and then to string so we can send it to a function drawText

drawText(tmpStr, coordinatesValue[4], color);
//Sends the text message to a function drawText to print it to
the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.0f", ((ceil((energy) / 10) * 10) * 3 / 4); //Inserts
the doubl into a statement and then to string so we can send it to a function drawText
```

```
drawText(tmpStr, coordinatesValue[5], color);
//Sends the text message to a function drawText to print it to
the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.0f", ceil((energy) / 10) * 10 / 2);
//Inserts the doubl into a statement and then to string so we can send it to a function drawText
```



```
drawText(tmpStr, coordinatesValue[6], color);  
//Sends the text message to a function drawText to print it to  
the screen
```

```
sprintf_s(tmpStr, (sizeof(tmpStr) / sizeof(tmpStr[0])), "%.0f", ceil((energy) / 10) * 10 / 4);  
//Inserts the double into a statement and then to string so we can send it to a function drawText  
drawText(tmpStr, coordinatesValue[7], color);  
//Sends the text message to a function drawText to print it to  
the screen
```

```
//free  
the malloc memory
```

```
for (int i = 0; i < (num_of_lines(txt) - 2); i++)  
//Loop that free the malloc memory that holds the gag data  
    free(gagDataMatrix[i]);  
    //free each of the malloc memory separately  
free(gagDataMatrix);  
    //free main the malloc memory  
free(allTimeAllEnergy);  
    //free the malloc memory that holds the time Coordinates  
free(allsize);  
    //free the malloc memory that holds the gag size and the distance between gags
```

```
glFlush();  
//flush GL buffers  
  
{  
  
//write txt using bitmap and stroke chars on the GL window(koby livne)  
//input-string (txt message)  
//output - nothing  
void drawText(const char *message, point messagePoint, float* color) }
```

```

        glColor3f(color[0], color[1], color[2]);          //the lines color (red)

        glRasterPos2f(messagePoint.x, messagePoint.y);

        while (*message)
//Keep running until all the text we want to insert into the GL window is finished

            glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *message++);    //write using bitmap and
stroke chars

        glEnd();
        //close the GL fancesen

    {

//calculate the distance between all points in a 2D matrix (koby livne)

//input - 2D (int) matrix the first cell contains the X coordinate of the point and the second cell contains the Y
coordinate And the fourth column holds the size of the Gag for For each row inside

//and a (float) pointers Which holds all the distances we calculated from the matrix

//output - a (float) pointers Which holds all the distances we calculated from the matrix

float * all_distance(int ** maxrix, float * size)

    }

    //A loop that passes on each row in a matrix and checks the distance between the coordinate stored inside

    for (int i = 0; i < gagToRemove * 2 - 1; i++)

    {

        if ((i + 1) % 2 == 0)

            //Checks if we calculate the distance between 2 gag edges now

            size[i] = (float)(distance(maxrix[i / 2], maxrix[i / 2 + 1]) - (sqrt(maxrix[i / 2][2]) / 2 +
sqrt(maxrix[i / 2 + 1][2]) / 2));    //Calculate the distance between 2 gag edges

            else

                //if we are not calculate the distance between 2 gag edges now

                size[i] = (float)sqrt(maxrix[i / 2][2]);

            //Calculate the gag size

```

```

    {

    return size;

//return a pointers Which holds all the distances between gags and the
gag size

{

//Prints a line on our graph every time between 2 points(koby livne)

//input -allPoint -All the points we need to print their line (pointer of point) ,pixels- number of pixels will be the
thickness of the line (int), color - The color you want(array of 3 floats)), numberOfPoints - numbers of points we need
to print (int(

//and a (double) pointers Which holds all the distances we calculated from the matrix

//output -nothing

void Prints_two_dots(point allPoint[], int pixels, int * color, int numberOfPoints)

{

    glLineWidth(pixels);
    //The number of pixels to use

    glColor3f(color[0], color[1], color[2]); //the lines color

    glBegin(GL_LINES);
        // The function we use in GL draws a line between two points

    for (int i = 0; i <numberOfPoints; i++)
        //Run until we print all the dots

    }

    glVertex2f(allPoint[i].x, allPoint[i].y); //A
position of one of the points

    {

    glEnd();

        //close the GL fancen

    {

//Prints a line on our graph given points(koby livne)

```

//input -allPoint -All the points we need to print their line (pointer of point) ,pixels- number of pixels will be the thickness of the line (int), color - The color you want(array of 3 floats)), numberOfPoints - numbers of points we need to print (int)

//and a (double) pointers Which holds all the distances we calculated from the matrix

//output -nothing

void Prints_a_number_of_points(point allPoint[], int pixels, float* color, int numberOfPoints)

}

glLineWidth(pixels); //The number of
pixels to use

glColor3f(color[0], color[1], color[2]); //the lines color

glBegin(GL_LINE_STRIP); // The function we use in
GL draws a line between several points in order

for (int i = 0; i <numberOfPoints; i++) //Run until we print all the dots

}

glVertex2f(allPoint[i].x, allPoint[i].y); //A
position of one of the points

{

glEnd(); //close the GL fancsen

{

//This function prints the menu and calls the needs functions for sections 5 (ori schwartz)

//In this section the user can choose a size of a square, the program will look for in the dust picture for the square with the largest total brightness

//The user can choose between 6 colors and 5 levels of brightness. The program will create a new BMP file of the dust picture in which the brightest square will be colored according to the user's request

//input - nothing

//output - nothing or 0 (int) in case the dust picture is too small

void section5()

}

int edge, maxEdge, minEdge, color, intensity;

int*dustiest;

```

INFOHEADER info;

FILE * original_pic;

fopen_s(&original_pic, "d2.bmp", "rb");

if (!original_pic) // checking if the file is
successfully opened

    exit; (0)

if(is_BMP(original_pic)) //calling the functions
"is_BMP" to check if the file a BMP file

}

info = read_Info(original_pic); //calling "read_Info
function to enter the information about the width, height and offset of the picture


//calculating the maximum and minimum size of the square that will be given to the user based on the height
and width of the picture

if (info.width <= info.height) maxEdge = info.width; //the smaller edge of the picture
defines the size of the square's edge

else maxEdge = info.height;

maxEdge = maxEdge / 3; //the maximum
size of the edge is a third of the samllest edge of the picture

maxEdge = maxEdge - (maxEdge % 10); //the remainder of 10 is
subtracted to give a round maximum edge size

minEdge = maxEdge / 3; //the minimum
size of the edge is a third of the samllest edge of the picture

minEdge = minEdge - (minEdge % 10); //the remainder of 10 is
subtracted to give a round minimum edge size


if (minEdge < 10) {

//if the minimum edge size is smaller the 10 pixels, a
notification that the picture is too small for this operation is printed

printf_s("\n\nThe picture is too small for this operation.\n\n"); //and the
function ends, returning 0

return; (0)

```

```

    {
do
    }

    printf_s("Please enter the edge size of a square between %d to %d\n", minEdge,
maxEdge); //printing a request for the size of the square
edge

    edge = input_control(minEdge, maxEdge);

//
calling "input_control" to allow scan from the user the edge size within the given limits

    { while (!edge; (
do}

    printf_s("choose a background color: \n1. blue\n2. red\n3. green\n4. cyan\n5. yellow\n6.
pink\n"); // printing a request for the color of the square

    color = input_control(1, 6);

// calling
"input_control" to allow scan from the user the edge size within the given limits

    { while (!color; (
do}

    printf_s("choose a intensity of the color: \n1. brightest\n2. brighter\n3. normal\n4.
darker\n5. darkest\n"); // printing a request for the brightness of the square

    intensity = input_control(1, 5);

// calling
"input_control" to allow scan from the user the edge size within the given limits

    { while (!intensity; (

    dustiest = dustiest_sqr(edge); //sending the
edge size to "dustiest_sqr" function to give the information about the square

//the information given is it's location, it's brightness and the number of equally bright squares

    printf_s("The center of the dustiest square is located at (%d,%d)\n", dustiest[0] + edge / 2, dustiest[1]
+ edge / 2); //printing the location of the middle of brightest square

    printf_s("the brightness ration (measured/maximum) is: %.2f%c\n", 100 * ((float)dustiest[2]) /
(edge*edge * 255), '%');
//printing how many square with equal value were found

```

```

        colored_square(dustiest[0], dustiest[1], edge, color, intensity);
//sending the information about the location of the dustiest square

//it's size and wanted color and
brightness to "colored_squar" function to create an BMP file

        free(dustiest); //freeing dustiest
array
    {
        fclose(original_pic); //closing the original picture file
    {
//This fuction locates the corner of the brightness square in the image. (ori schwartz)
//The function calls the matrix creation function and finds the square using 4 loops.
//the 2 inner loops sums the brightness of the square and the 2 outer loops moves the point where the sums starts
within the matrix
//and compares the information of each square to find brightest one
//input - (int) the size of the edge of the square
//output - (int array pointer) the information of the location the brightness of the square, and how many equally bright
squares were found

int* dustiest_sqr(int edge)
}

    int* dustiest; //dustiest [0] - coordinate
x, dustiest [1] - coordinate y, dustiest [2] - total square brightness

    dustiest = malloc(sizeof(int) * 3); //declaration an int array using malloc so it's
information wont be lost when the function end .

    dustiest[2] = 0; //giving a starting brightness value
of 0

    pix** matrix;

    int i, j, x, y, padding, brightness = 0;

    INFOHEADER info;

    FILE* picf;

    fopen_s(&picf, "d2.bmp", "rb");

    if (!picf) // checking if the file is successfully opened

```

```

        exit;(0)

        is_BMP(picf);                                //checking if the file is BMP

        info = read_Info(picf);                      //calling "read_Info function to enter the information about the
width, height and offset of the picture

        padding = 4 - info.width % 4;                //calculate the amount of padding

        if (padding == 4) padding = 0;

        matrix = Matrix_Creation(picf, info.width, info.height, info.offset); //creating a matrix of pixels structure
(struct pix)

// the next 4 loops calculates the total brightness of the square of the
given size

// the outer 2 loops move the starting location of the square to different
places in the matrix and compares the square's brightness

// the inner 2 loops sums the brightness value for each square

for (y = 0; y<info.height - edge; y++)                //moves the starting place along y axis
}

for (x = 0; x < info.width - edge; x++)                //moves the stating position along x axis
}

for (j = y; j < edge + y; j++)                        //move the line that is being
summed from the starting position till the end of the edge along y axis
}

for (i = x; i < edge + x; i++)                        //sums the brightness of a line along
x axis, starting from the starting position to the end of the edge size;
}

        brightness = brightness + matrix[j][i].power;    //sums the total
brightness of the square

```



```

        {
            {
                if (dustiest[2] < brightness) {           //if the currently summed square is brighter
than the brightest square that was found before:
                    dustiest[0] = x;                     //the x coordinate of the square is
saved
                    dustiest[1] = y;                     //the y coordinate of the square is
saved
                    dustiest[2] = brightness;           //the total brightness the square is saved

            {
                brightness = 0;                          //total brightness is zeroed
before calculating the next square
            {
                fclose(picf);
                for (i = 0; i < info.height; i++)
            }

                free(matrix[i]);    //memory release

            {
                free(matrix);

                return(dustiest);    //the information of the dustiest square is returned
        {
//This function uses the original dust pic, and the information of the dustiest square to create a new BMP FILE
        (ori schwartz)

//The function changes the value of specific bits in the pixels that are included in the dustiest square according to
the user's color and brightness choice

//input - (int x1) the x coordinate of the dustiest square,(int y1) the y coordinate of the dustiest square

//input - (int size) the edge's size of the square, (int choice) the color of the square, (int intensity) the intensity of the
color

//output - nothing

void colored_square(int x1, int y1, int size, int choice, int intensity)

```

```

}

INFOHEADER info;                                //holds the picture information

int i, j, k;                                     //loop
indexes

    unsigned int get_char, brightness = 255, padding;    //get_char uses for coping 1 byte
from the original picture to the new picture, brightness holds the brightness volume, padding holds the padding value

    FILE * original_pic;                           //pointer for the original picture

    fopen_s(&original_pic, "d2.bmp", "rb");           //open for reading

    if (!original_pic)                               // checking if the file is successfully opened
        exit; (0)

    is_BMP(original_pic);                           //checking if the file is a BMP file

    FILE * copied_pic;                               //pointer for the new
picture

    fopen_s(&copied_pic, "coloredDust.bmp", "wb");     //open for writing

    if (!copied_pic)                                 // checking if the file is successfully opened
        exit; (0)

    info = read_Info(original_pic);                   //calling "read_Info
function to enter the information about the width, hieght and offset of the picture

    padding = info.width % 4;                         //calculating the padding
of the image

    switch (intensity)                                //adjusting the brightness value of
the color
    {
        case 1: break;

        case 2: {brightness -= 50; break; }

        case 3: {brightness -= 100; break; }

        case 4: {brightness -= 150; break; }
    }

```

```

case 5: {brightness -= 200; break; }

{
//offset coping-a loop which write the information of the picture to the new file
fseek(original_pic, 0, SEEK_SET);
for (i = 0; i < info.offset; i++)
}

get_char = fgetc(original_pic);           //copy the information form the original picture into
the new one

fputc(get_char, copied_pic);

{
//copy and paint
for (i = 0; i < info.height; i++)           //This loop goes over each line in the picture and
create the image that is needed
}

for (j = 0; j < (info.width); j++)           //this loop goes over each pixel in the line and insert
the information that is needed
}

get_char = fgetc(original_pic);           //get the current byte of the pixel is saved
into get_char

if ((x1 <= j) && (j <= (x1 + size)) && (i >= y1) && (i <= (y1 + size)))

//if the pixel is inside the square then the information that is inserted to the file is
changed

//according to the user color Preferences
}

switch (choice)
{
case 1:           //if the user chose blue, the 1st bit gets the value of
'brightness' and the other 2 are copied from the original picture
}

```

```

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (blue byte)

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        break;

    {

        case 2: //if the user chose red, the 3rd byte gets the value of
'brightness' and the other 2 are copied from the original picture

    }

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (red byte)

        break;

    {

        case 3: //if the user chose green, the 2nd byte gets the value of
'brightness' and the other 2 are copied from the original picture

    }

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

```

```

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (green byte)

        fseek(original_pic, 1, SEEK_CUR);
//move original picture pointer 1 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        break;

    {

        case 4: //if the user chose cyan, the 1st and 2nd bits get the
value of 'brightness' and the other one is copied from the original picture

    }

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (blue byte)

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (green byte)

        fseek(original_pic, 2, SEEK_CUR);
//move original picture pointer 2 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        break;

    {

        case 5: //if the user chose yellow, the 2nd and 3rd bits get the
value of 'brightness' and the other one is copied from the original picture

    }

        fseek(original_pic, 2, SEEK_CUR);
//move original picture pointer 2 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (green byte)

```

```

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (red byte)

        break;

    {

        case 6: //if the user chose pink, the 1st and 3rd bits get the
value of 'brightness' and the other one is copied from the original picture

    }

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (blue byte)

        fseek(original_pic, 2, SEEK_CUR);
//move original picture pointer 2 byte

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture

        fwrite(&brightness, 1, 1, copied_pic);
//put the set brightness (red byte)

        break;

    {

        default:
//in other cases- break

        break;

    {

    {

        else { //if the pixel is outside the square 3 bit are copied
from the original picture into the new one

            fputc(get_char, copied_pic);
//copy the copied byte to the new picture

            get_char = fgetc(original_pic);
//get the current byte of the pixel is saved into get_char

            fputc(get_char, copied_pic);
//copy the copied byte to the new picture

            get_char = fgetc(original_pic);
//get the current byte of the pixel is saved into get_char

```

```

        fputc(get_char, copied_pic);
//copy the copied byte to the new picture
    {
        {
            for (k = 0; k < padding; k++)
                //at the end of each pixel line, the bit of the padding is
copied
                //from the
original picture to the new one,
            }
            get_char = fgetc(original_pic);
            //get the current
byte of the pixel is saved into get_char
            fputc(get_char, copied_pic);
            //copy the copied
byte to the new picture
        {
            {
                fclose(original_pic);
                //closes the original picture
                fclose(copied_pic);
                //closes the new picture
            }
        }
    }

```

תיאור תהליך סעיף 4

1. משתמש בוחר בסעיף 4
2. נכנס ל `Builds_the_graph`
3. בודק את מספר השורות בקובץ כדי לבדוק אם יש פחות גאגים מ9 ומדפיס הודאה מתאימה
4. משתמש בוחר כמות גאגים לסילוק
5. מהתחלים את הפרמטרים הבסיסיים של חלון GLn (גודל, סגנון, מיקום התחלתי)
6. נכנס ל `display`
7. בונה את הגרפיקה שלא מצריכה חישובים (הצירים והקווים של השנתות על ידי שליחה לפונקציות `Prints_a_number_of_points` אשר מדפיסה קוים על הגרף כאשר שולחים לה נקודות)
8. מהתחל את המשתנים בשביל החישובים
9. מהתחל את המבני הנתונים שלי (ראה הסבר על כל אחד מהם בנפרד בסוף הפסדו)
10. קורא את הנתונים מהקובץ `dust` מכניס אותם ל `gagdatamatrix`
11. נכנס ל `all_distance` מחשב את כל המרחקים בין גאגים וגדלים של הגאגים ומכניס אותם ל `allsize` (דוגמת חישוב בסוף המסמך)
12. מאתחל את הצעד זמן והאנרגיה התחלתית שלי ל0
13. לולאה חיצונית שרצה עד שהיא מגיעה לבחירה של המשתמש כמה גאגים להעיף כפול 2 - 1
- 13.1. לולאה פנימית שרצה על כל אחד מהתאים ב `allsize` הופכת אותם לזמן (בעזרת הנוסחה $x=v*t$) ומסיימת כאשר היא מגיעה לגודל הזמן שמאוחסן בתא מתקדמת בצעדי זמן של 0.001
- 13.1.1. חישוב של האנרגיה שבזבזנו עד עכשיו וקידום צעד זמן (דוגמת חישוב בסוף המסמך)
- 13.1.2. בדיקה האם הנגזרת של האנרגיה בצעד זמן הנוכחי גדולה מ550 (דוגמת חישוב בסוף המסמך)
- 13.1.2.1. (אם כן) שמירה של המיקום הנוכחי עם כל הפרמטרים
- 13.2. הכנסת האנרגיה הנוכחית והזמן הנוכחי ל `alltimeallenargy` (אנחנו נקודת מפנה בחישוב האנרגיה בדיוק יצאנו מהלולאה שמחשבת את הזמן שעברנו עד סיום המרחק הנוכחי נדרש להדפיס נקודה זו על הגרף)
14. לולאה חיצונית שרצה עד שהיא מגיעה לבחירה של המשתמש כמה גאגים להעיף כפול 2 - 1
- 14.1. שולח כל פעם 2 נקודות ל `Prints_two_dots` וככה מדפיס את הגרף שלנו
15. בודק האם `error` לא ריק (כלומר הנגזרת הייתה גדולה מ550)
- 14.1 (אם כן) מדפיס את ההודאה במקום שהיא קרתה על גבי הגרף בעזרת שליחה ל `drawText`

16. מדפיס את השנתות של ציר הזמן והאנרגיה כאשר השתנתה הגדולה היא הזמן המקסימלי מעוגל למעלה וכל שנתה קטנה יותר קטנה מהקודמת ברבע בעזרת drawText

17. משחרר את כל הזיכרון שהוקצה malloc בתוכנה

18. מסיים את display וחוזר לBuilds_the_graph

19. נכנס לmyKey ומאזין האם נלחץ מקש מהמקלדת

18.1 אם כן סוגר את התוכנה ואת כל מה שקשור אליה

20. נכנס לglutMainLoop ומצייר לנו את הגרף על המסך (תוך קריאות התפעלות מהצד שרואה את הגרף קם לתחייה)

תיאור שמירת הנתונים הנתונים:

בסעיף 4 אנו משתמשים ב4 מבני נתונים שלכל אחד מהם תפקיד שונה (3 מאותחלים באמצאות malloc ו1 זה מבנה)

Point- מבנה פשוט של נקודה מכיל קורדינטה X וקורדינטה Y.

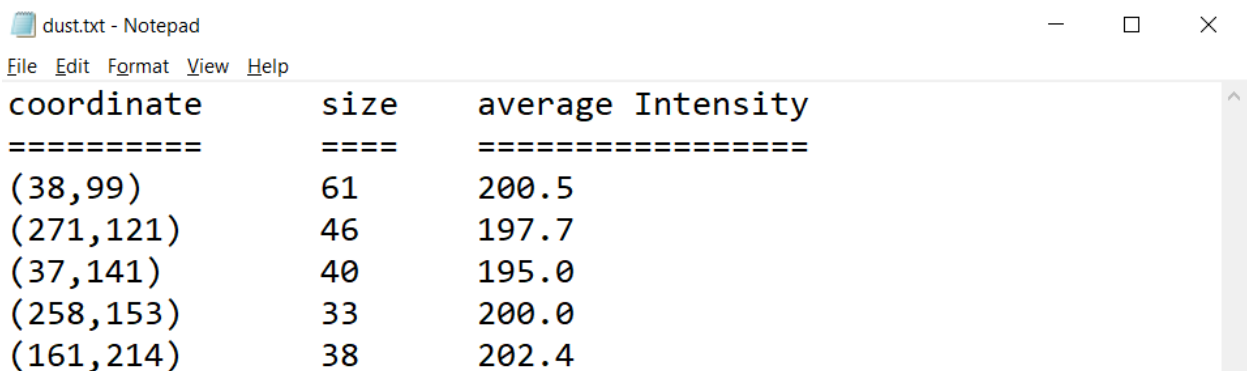
Gagdatamatrix-מבנה נתונים בתצורה של טבלה דו מימדית שגודלה 3 על מספר הגאגים בקובץ הנתונים נלקחים מהקובץ dust כאשר כל שורה מייצגת גאג אחד לפי הסדר בקובץ.

עמודה 1 : קורדינטה X של הגאג

עמודה 2 : קורדינטה Y של הגאג

עמודה 3 : גודל הגאג

דוגמה של המבנה נתונים gagdatamatrix בסעיף 4 בבחירה של סילוק 3 גאגים :



coordinate	size	average Intensity
=====	====	=====
(38,99)	61	200.5
(271,121)	46	197.7
(37,141)	40	195.0
(258,153)	33	200.0
(161,214)	38	202.4

gagdatamatrix		
גודל הגאג	קורדינטה Y של הגאג	קורדינטה X של הגאג
61	99	38
46	121	271
40	141	37

Allsize - מבנה נתונים בתצורת רשימה שמכיל את הגדלים של הגאגים והמרחק בניהם הגודל שלו יהיה כמות הגאגים כפול 2-1 שמסלקים את הנתונים אנחנו מקבלים מ מהקורדינטות שב Gagdatamatrix אשר עוברות חישוב של מרחקים ושורש בפונקציה all_distance לבסוף ישנו רשימה אשר מסודרת כך : במקומות האי זוגיים ישנם הגודל של הגאג לפי הסדר הופעה שלהם ובמקומות הזוגיים ישנם המרחקים בין הגאגים (בגאג שמימינו ומשמאלו)

דוגמה עם אותם נתונים כמו לפני זה :

allsize				
גאג שלישי: 6.32	מרחק בין גאג שני לשלישי: 228.3	גאג שני: 6.78	מרחק בין גאג ראשון לשני: 226.74	גאג ראשון: 7.81

alltimeallenargy - מבנה נתונים בתצורת רשימה אשר מכיל טיפוס מסוג point אשר מחזיק קורדינטה X וקורדינטה Y. הגודל שלו יהיה כמות הגאגים שמסלקים כפול 2. את הנתונים אנו מקבלים מתוך החישוב של המשוואה וההתקדמות של הזמן, כאשר אנו צריכים להדפיס נקודה על הגרף היא נשמרת בתוך הרשימות הללו (קורדינטה הזמן נקראת X וקורדינטה האנרגיה נקראת Y) לאחר מכן אנו מדפיסים את הנקודות שבפנים וכך נוצר הגרף (הם מאותחלים עם 0 באיבר הראשון כי אנו מתחילים מזמן ואנרגיה שווה ל0)

דוגמה עם אותם נתונים כמו לפני זה :

alltimeallenargy					
אחרונה X: 4.762	רביעית X: 4.698	שלישית X: 2.415	שניה X: 2.347	ראשונה X: 0.079	התחלה X: 0
אחרונה Y: 78.01	רביעית Y: 57.08	שלישית Y: 34.25	שניה Y: 23.68	ראשונה Y: 1.001	התחלה Y: 0

דוגמת חישוב של חישוב האנרגיה :

$$E[i+1] = (E[i]P[i]S[i]A + VB) * \Delta t + E[i] : \text{הנוסחה אחרי שסידרנו אותה}$$

כאשר :

$$\Delta t = 0.001$$

$$VB = 0.1 * 100 = 10$$

$$A = 0.75$$

$S[i]$ = גודל המרחק / גאג שאנחנו מעליו

$P[i] = 1$ אם אנחנו מעל גאג 0 אם אנחנו לא מעל גאג

$E[i] = 0$ = האנרגיה בתחילת צעד הזמן

דוגמת חישוב לחישוב המרחקים :

חישוב גודל גאג יתבצע באמצעות שורש של מספר הפיקסלים שלו :

$$size = \sqrt{31} = 5.568 \text{ : ולכן הגודל שלו יהיה}$$

דוגמת חישוב למרחק בין קווצת של גאגים :

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - \frac{\sqrt{size_1}}{2} - \frac{\sqrt{size_2}}{2} : \text{יתבצע בעזרת הנוסחה}$$

גאג ראשון בעל 61 פיקסלים ומרכז (38,99)

גג שני בעל 46 פיקסלים ומרכז (271,121)

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - \frac{\sqrt{size_1}}{2} - \frac{\sqrt{size_2}}{2} = \sqrt{(38 - 271)^2 + (99 - 121)^2} - \frac{\sqrt{61}}{2} - \frac{\sqrt{46}}{2}$$

$$D = 226.74$$

דוגמת חישוב מעל הגאג הראשון :

$$E[i] = 0.01, P[i] = 1, S[i] = \sqrt{31}$$

$$E[i+1] = (E[i]P[i]S[i]A + VB) * \Delta t + E[i] \rightarrow (0.01 * 1 * \sqrt{31} * 0.75 + 10) * 0.001 + 0.01$$

$$E[i+1] = 0.02$$

דוגמת חישוב בין גאגים :

$$E[i] = 1.29, P[i] = 0, S[i] = 123.49$$

$$E[i+1] = (E[i]P[i]S[i]A + VB) * \Delta t + E[i] \rightarrow (1.29 * 0 * 123.49 * 0.75 + 10) * 0.001 + 1.29$$
$$E[i+1] = 1.3$$

דוגמת חישוב של בדיקת הנגזרת :

בדיקת הנגזרת תתבצע ב אמצעות :

$$\frac{E[i+1] - E[i]}{\Delta t} > 550$$

לכן בשביל צעד הזמן: $E[i] = 1.29, \Delta t = 0.001, E[i+1] = 1.3$

$$\frac{E[i+1] - E[i]}{\Delta t} = \frac{1.3 - 1.29}{0.001} = 10 < 550$$

לכן בצעד זמן זה לא חרגנו מהמותר ולא נדרש להקפיץ הודעת שגיאה

אופן פעולת סעיף 5:

סעיף 5 אחראי לעבור על התמונה הנתונה לחפש את ריכוז הבהירות הכי גדול ומצייר ריבוע בצבע ובהירות נבחרת באזור זה.

כאשר המשתמש לוחץ 5, הוא מתבקש לבחור את גודל הריבוע (לפי צלע) הרצוי בין טווח נתון (משתנה בהתאם לגודל התמונה). לאחר מכן המשתמש בוחר את צבע הריבוע מתוך 6 אפשרויות, לאחר מכן את רמת הבהירות מתוך 5 רמות בהירות שונות. התוכנה תיצור את הריבוע ותבדוק את רמת הבהירות בתוכו, לאחר מכן תזיז אותו פיקסל אחד הצידה ושומרת את רמת הבהירות של הריבוע שרמת הבהירות שלו היא הגבוהה ביותר. לאחר שעברה על כל התמונה היא יוצרת עותק שלה וצובעת את הריבוע בעל הבהירות המקסימלית בצבע וברמת הבהירות שנבחרה. לבסוף התוכנה שומרת את התמונה החדשה עם הריבוע הצבוע בשם coloredDust ומדפיסה למשתמש את מרכז הריבוע ואת אחוז הבהירות ביחס לשטח עם בהירות מקסימלית.

דוגמאות הרצה:

מסך פתיחה:

```
C:\WINDOWS\system32\cmd.exe

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice:
```

לחיצה על אפשרות 1:

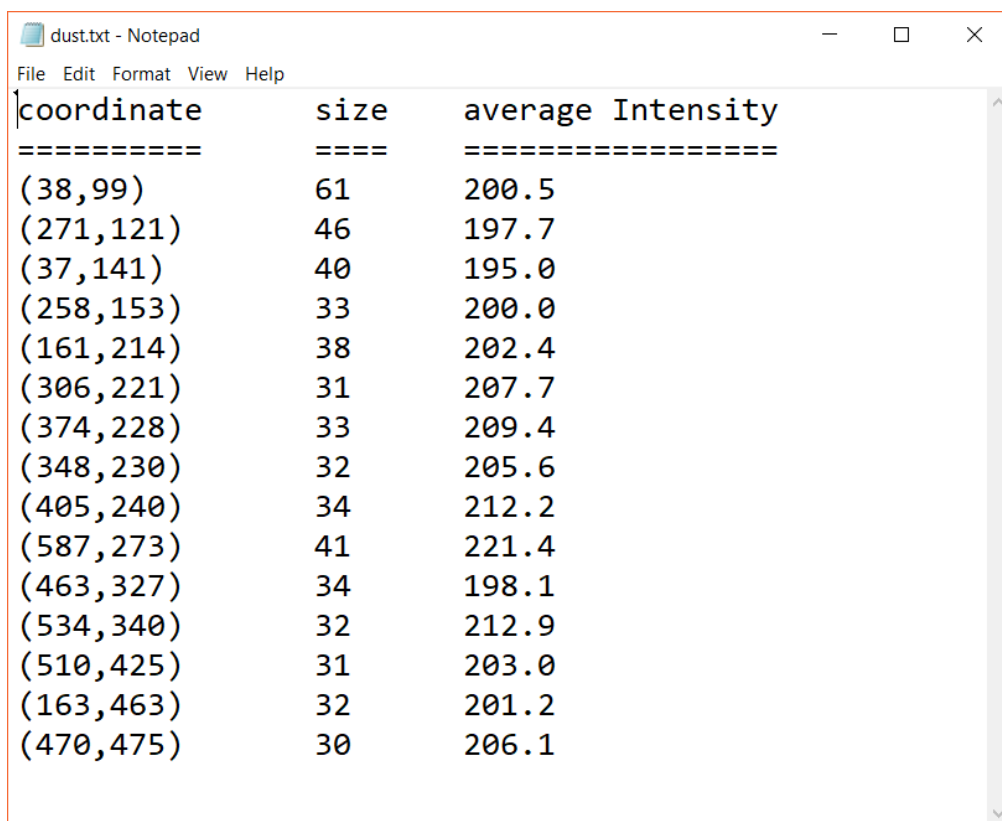
```
C:\WINDOWS\system32\cmd.exe

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice: 1

cordinates are (38,99)
size 61
Average intensity 200.5
Total of 15 big dust particles.

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice:
```

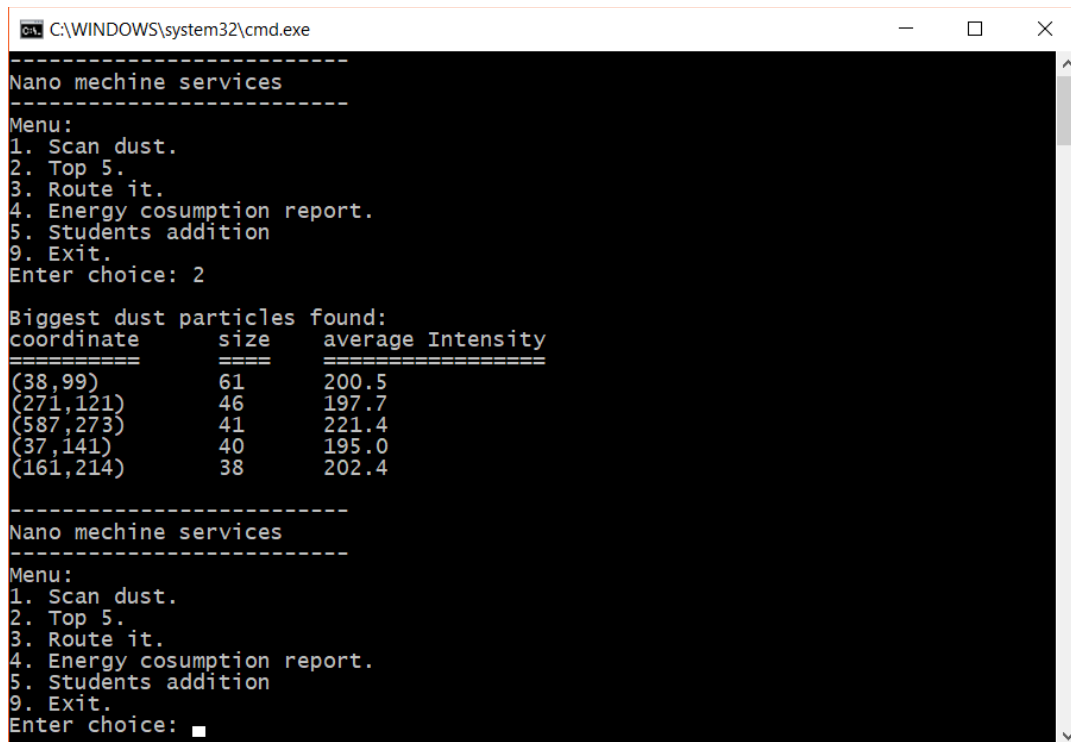
הקובץ dust שנוצר



The screenshot shows a Notepad window with the following table content:

coordinate	size	average Intensity
(38,99)	61	200.5
(271,121)	46	197.7
(37,141)	40	195.0
(258,153)	33	200.0
(161,214)	38	202.4
(306,221)	31	207.7
(374,228)	33	209.4
(348,230)	32	205.6
(405,240)	34	212.2
(587,273)	41	221.4
(463,327)	34	198.1
(534,340)	32	212.9
(510,425)	31	203.0
(163,463)	32	201.2
(470,475)	30	206.1

כאשר לוחצים על אופציה 2:



```
C:\WINDOWS\system32\cmd.exe

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice: 2

Biggest dust particles found:
coordinate      size      average Intensity
=====
(38,99)         61        200.5
(271,121)       46        197.7
(587,273)       41        221.4
(37,141)        40        195.0
(161,214)       38        202.4

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice: █
```

כאשר לוחצים על 3 ובוחרים את המספר 7:

```
C:\WINDOWS\system32\cmd.exe
-----
Nano mechine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice: 3

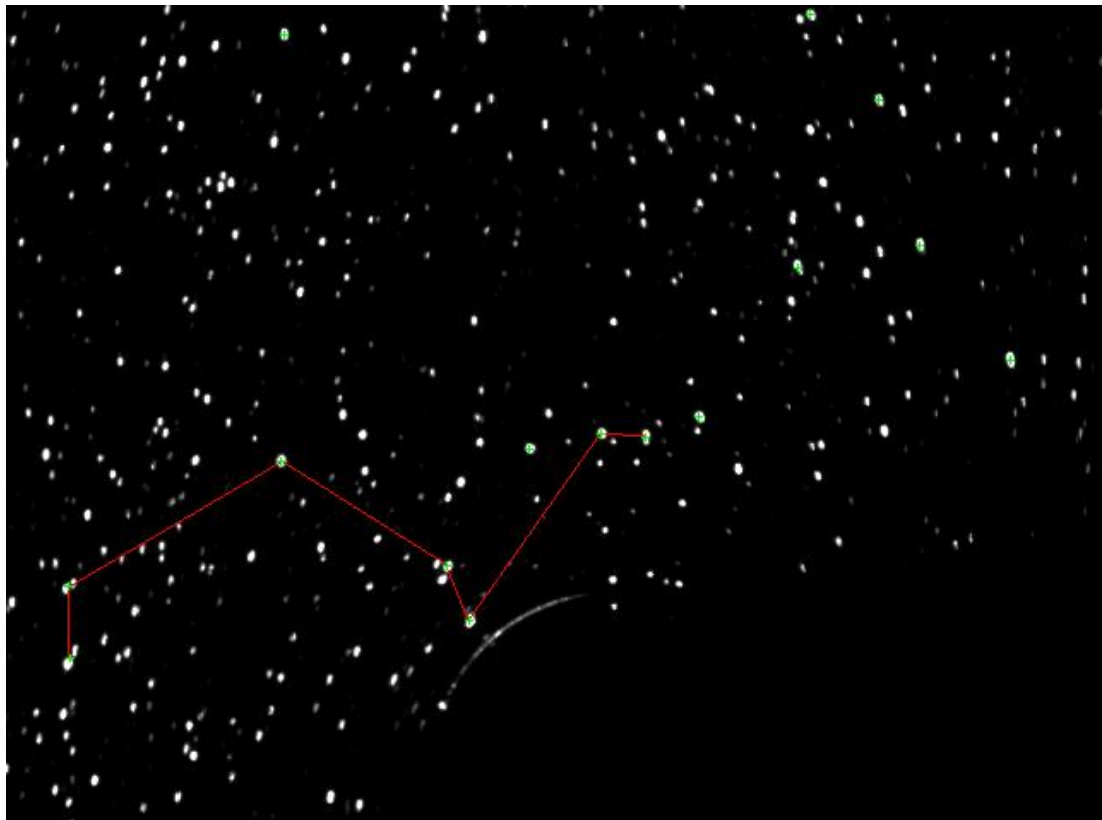
First map connection:
(38,99) neighbours are:(258,153),(161,214),(37,141)

Select a number between 1 to 15: 7
The selected number is 7
Best route found:
(374,228)->(348,230)->(271,121)->(258,153)->(161,214)->(37,141)->(38,99)

Check for a new copy file dustcopy.bmp
Done connecting biggest dust particles in dustcopy.bmp
Done marking coordinate of biggest dust particles in dustcopy.bmp

-----
Nano mechine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy cosumption report.
5. Students addition
9. Exit.
Enter choice: █
```

התמונה עם המסלול הקצר ביותר שסעיף 3 יצר:



כאשר בחרנו את 4:

```
C:\WINDOWS\system32\cmd.exe
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy consumption report.
5. Students addition
9. Exit.
Enter choice: 3

First map connection:
(38,99) neighbours are:(258,153),(161,214),(37,141)

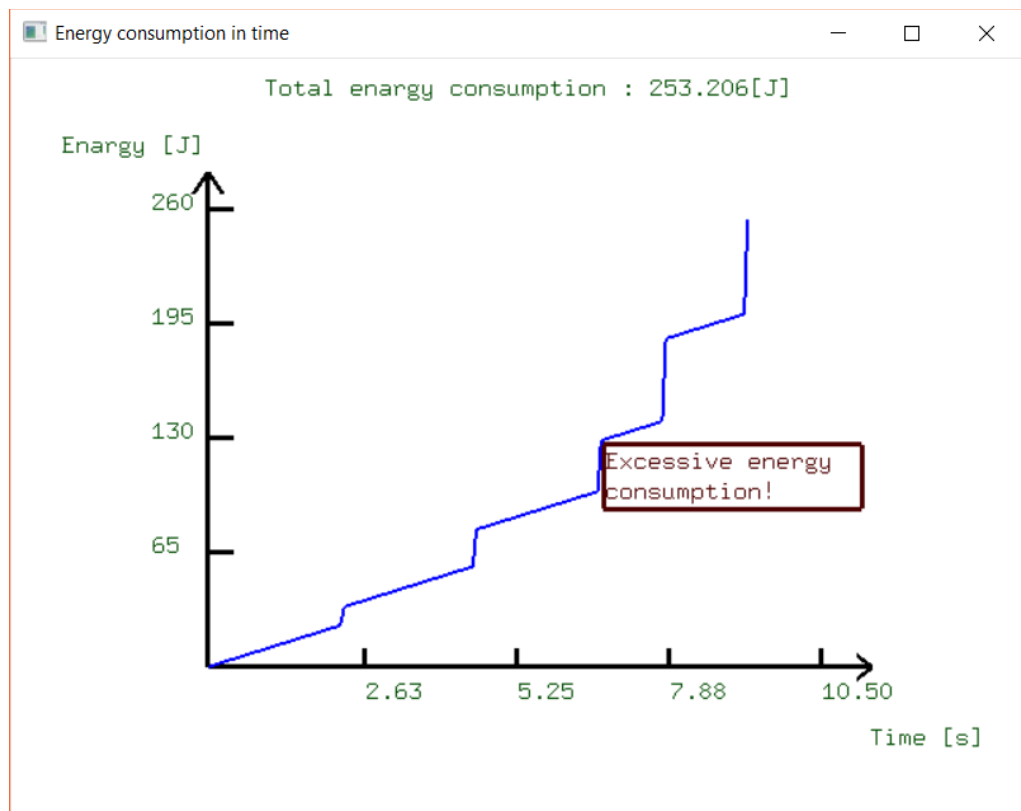
Select a number between 1 to 15: 7
The selected number is 7
Best route found:
(374,228)->(348,230)->(271,121)->(258,153)->(161,214)->(37,141)->(38,99)

Check for a new copy file dustcopy.bmp
Done connecting biggest dust particles in dustcopy.bmp
Done marking coordinate of biggest dust particles in dustcopy.bmp

-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy consumption report.
5. Students addition
9. Exit.
Enter choice: 4

Enter length of route up to 9: 6
```

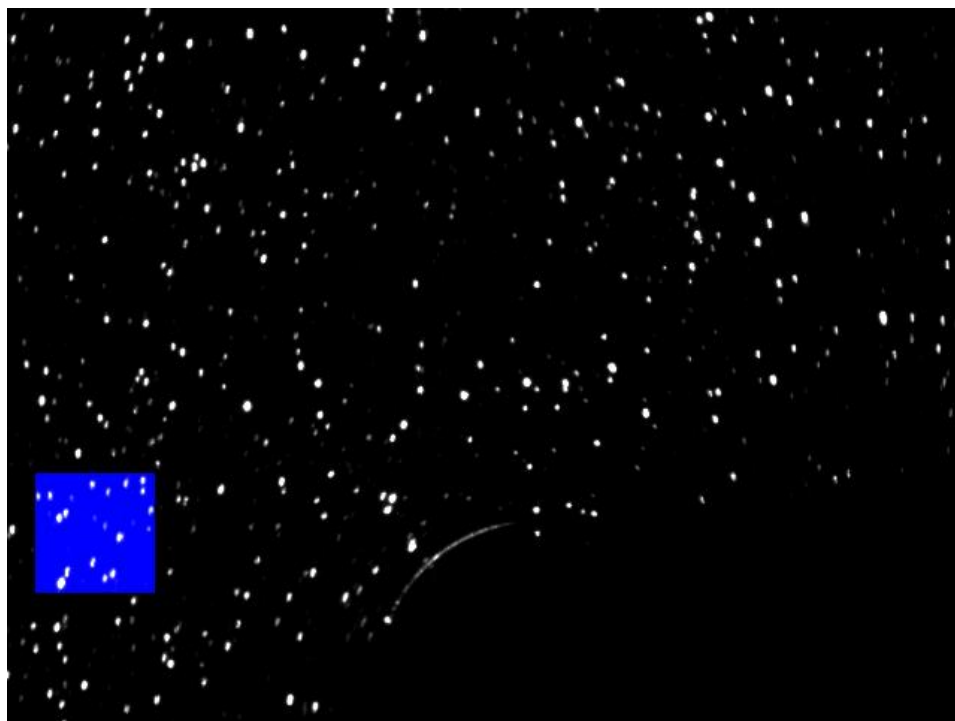
הגרף שנוצר עבור סילוק של 6 גאגים:



בחירה של 5:

```
C:\WINDOWS\system32\cmd.exe
-----
Nano machine services
-----
Menu:
1. Scan dust.
2. Top 5.
3. Route it.
4. Energy consumption report.
5. Students addition
9. Exit.
Enter choice: 5
Please enter the edge size of a square between 50 to 160
79
choose a background color:
1. blue
2. red
3. green
4. cyan
5. yellow
6. pink
1
choose a intensity of the color:
1. brightest
2. brighter
3. normal
4. darker
5. darkest
1
The center of the dusstiest squar is located at (486,395)
the brightness ration (measured/maximum) is: 4.25%
```

התמונה שנוצרה (הריבוע עם הכי הרבה בהירות בתמונה) כאשר בחרנו גודל של 79 צבע כחול הכי בהיר:



חלוקת עבודה:

החלוקה בגדול הייתה :

עומר סעיף 1 , אורי סעיף 2 +5+עזרה ב3, אלעד סעיף 3 ,יעקב סעיף 4 ודוח.
בפועל כולנו השתתפנו בכתיבת כל הסעיפים ועזרנו אחד לשני לאורך כל הדרך.

*כלל המקורות בעבודה נלקחו מהרצאות והתרגולים אשר נמצאים באתר המודל.